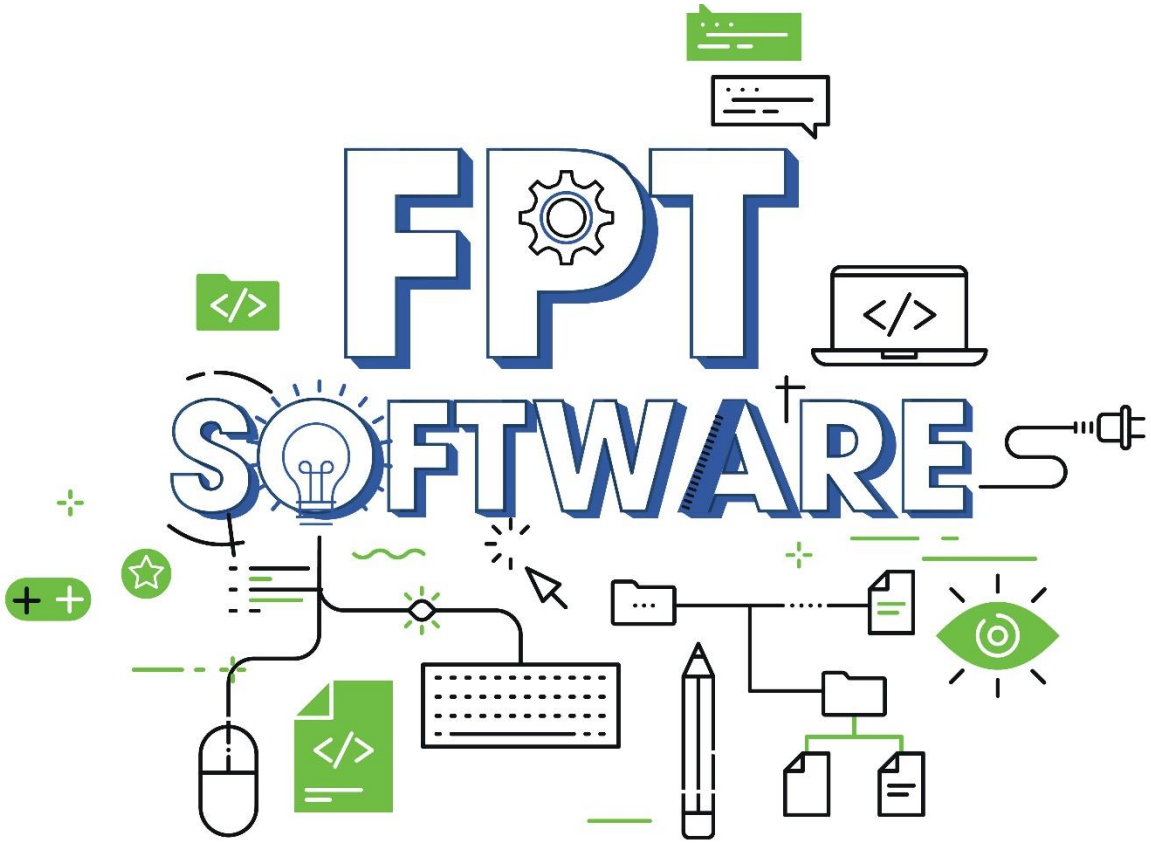
Java Strings

Instructor: DieuNT1



Agenda

01. Creating Strings

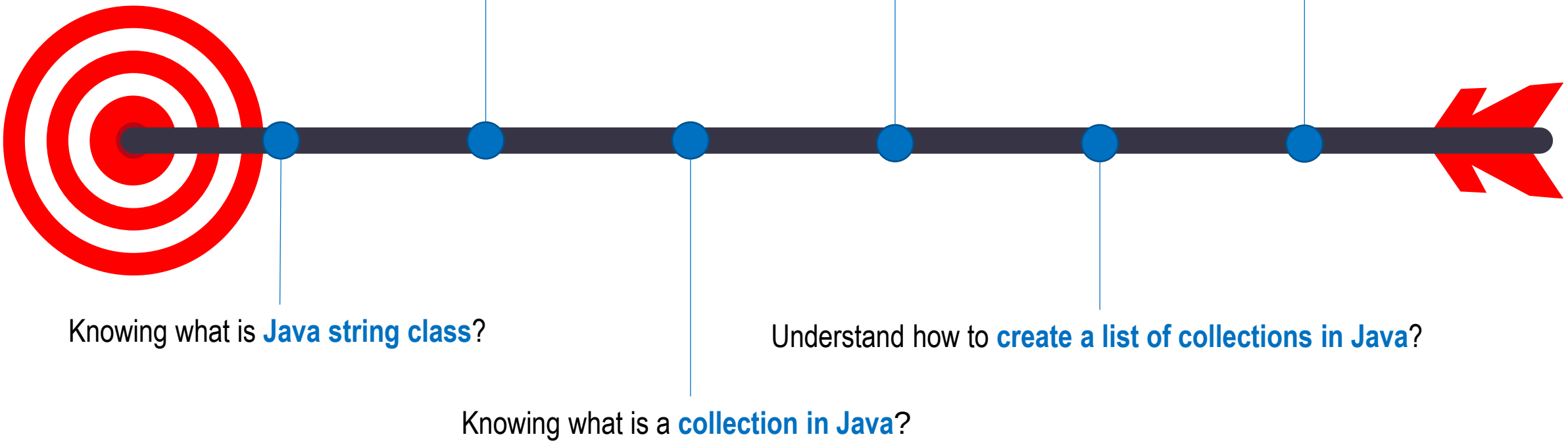
02. String Immutable

03. String Methods

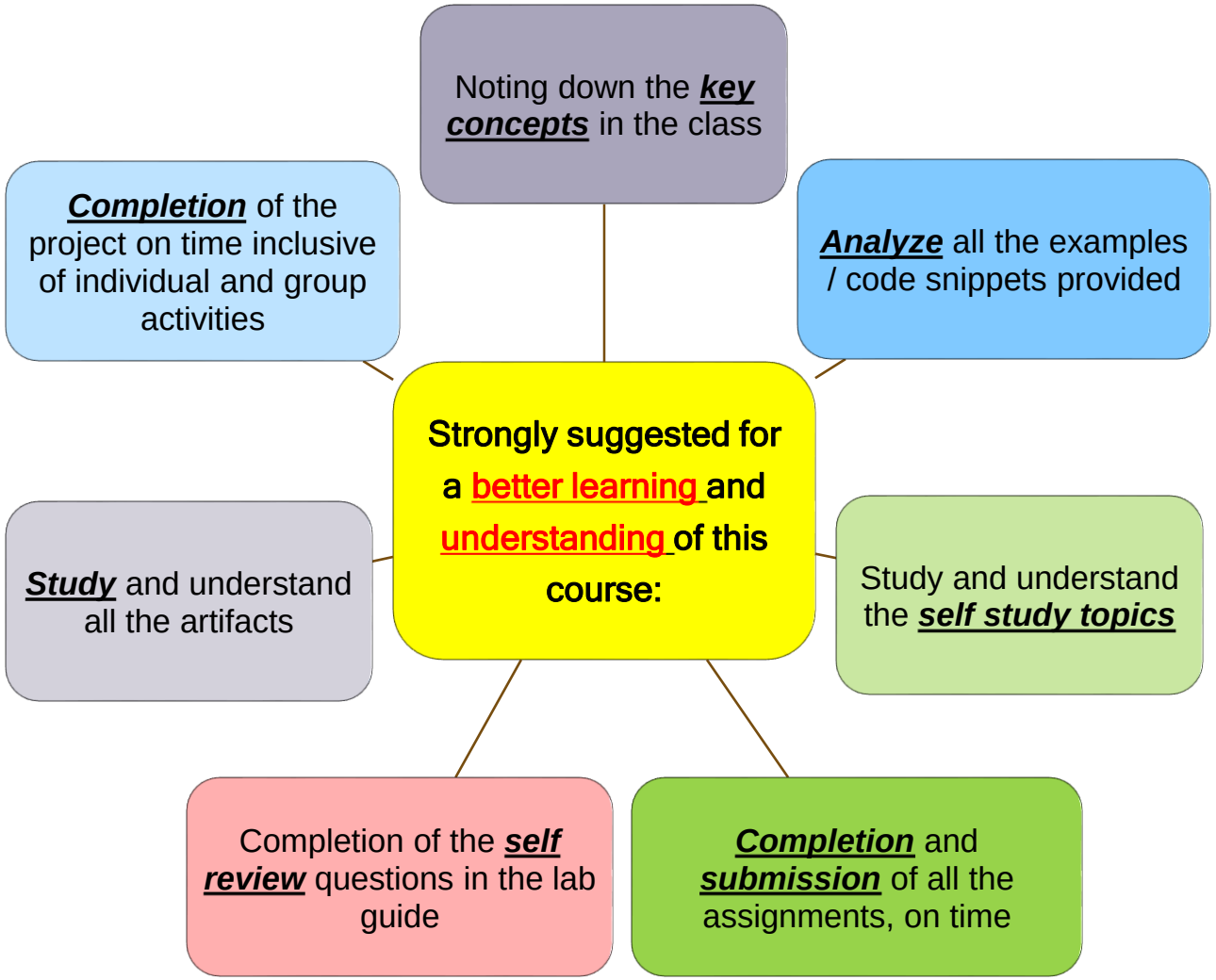
04. StringBuilder and StringBuffer

05. Regular Expression

Lesson Objectives



Learning Approach



Section 1

Creating Strings

String class

String

- Is a sequence of characters, for e.g. “Hello” is a string of 5 characters.

In Java, string

- Is an **immutable object** which means it is **constant** and can **cannot be changed** once it has been created.

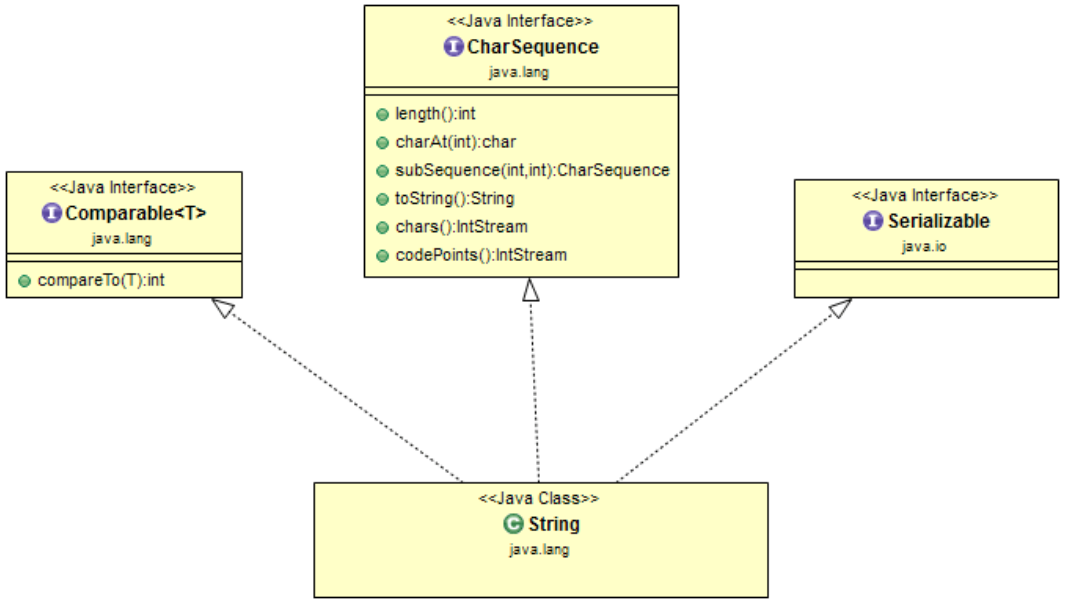
Creating a String

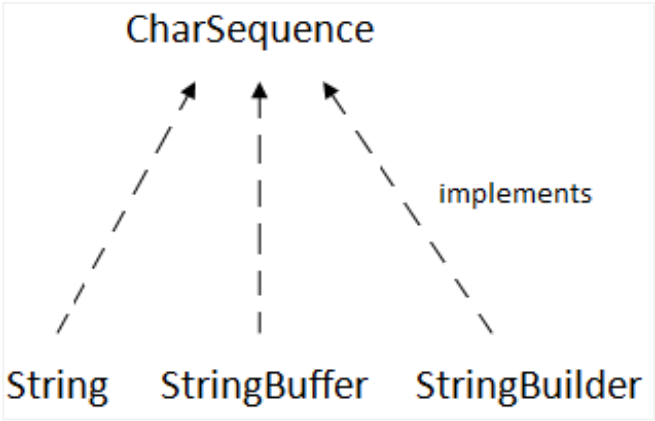
- There are **two ways** to create a String in Java
  - String **literal**
  - Using **new** keyword

String class hierarchy

What is a Java String? In Java, a string is an object that represents a sequence of characters or char values. The *java.lang.String* class is used to create a Java string object.

- The **String** class implements **Serializable**, **Comparable** and **CharSequence** interfaces.

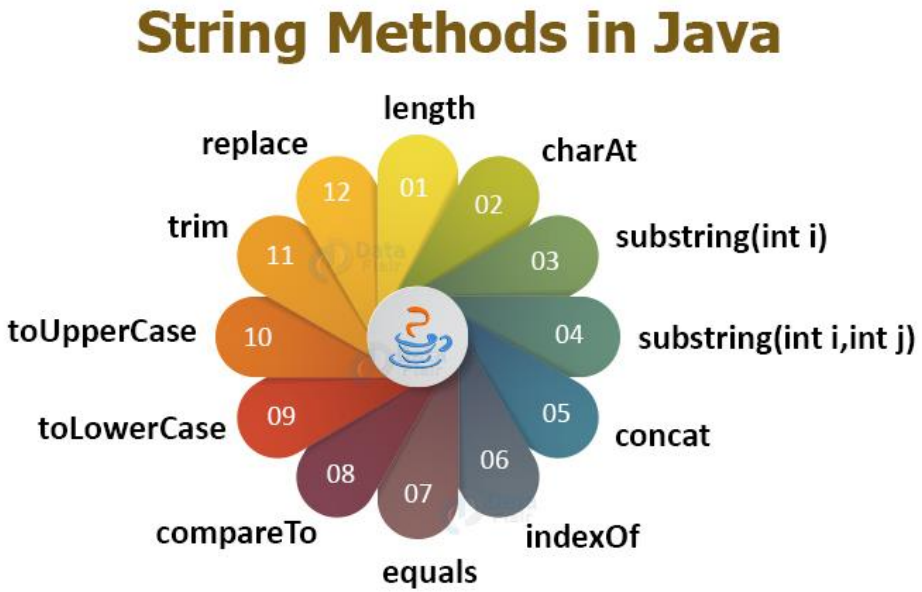




- The **CharSequence** interface is used to represent the sequence of characters. **String**, **StringBuffer** and **StringBuilder** classes implement it.

The String class

- **String** class has 11 constructors, 40+ methods
- Very useful for programming and learning OOP



Constructing a String

- **There are two ways to create a String object:**

✓ **By string literal** : Java String literal is created by using double quotes.

```
String s = "Welcome to Java";
```

✓ **By new keyword** : Java String is created by using a keyword “new”. **Can create String from *string literal* or *char[]***

- *String literal is characters in quotes.* Java treats string literals as String objects

```
String message = new String("Welcome to Java");
```

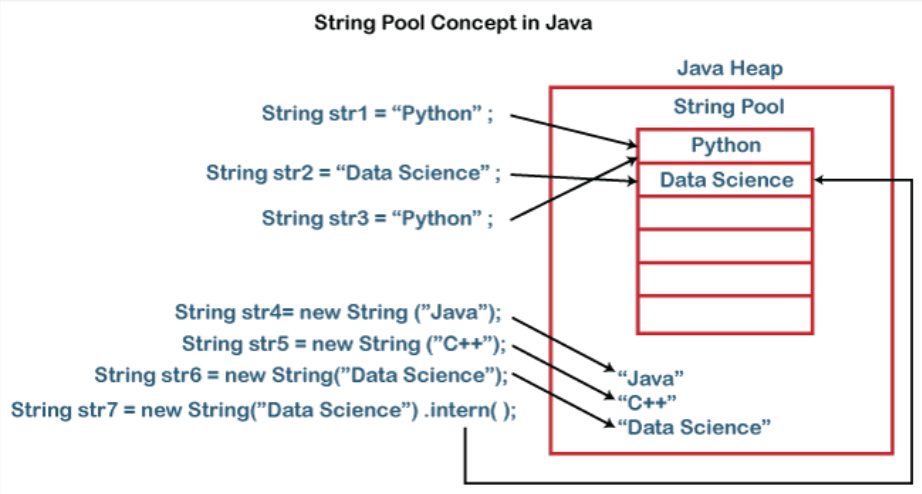
- *Create from char[]:*

```
char[] chars = {'G', 'o', ... };
String s = new String(chars);
```

Java String pool

Java String pool refers to collection of Strings which are stored in heap memory that is used to store string literals.

- **String literals** are created by enclosing a sequence of characters within double quotation marks, such as "Hello, World!".
- Whenever a **new object is created**, String pool first checks whether the object is already present in the pool or not:
  - ✓ **If it is present**, then same reference is returned to the variable
  - ✓ Else new object will be created in the String pool and the respective reference will be returned.



Java String pool

- For example, if you have multiple variables initialized with the same string literal, they will all refer to *the same string object in the string pool*:

```
String str1 = "Hello";
String str2 = "Hello";
System.out.println(str1 == str2); // Output: true
```

- If you create a string object using the new keyword, it will not be interned and will not be part of the string pool:

```
String str3 = new String("Hello");
String str4 = new String("Hello");
System.out.println(str3 == str4); // Output: false
```

- You can explicitly intern a string object using the **intern()** method.

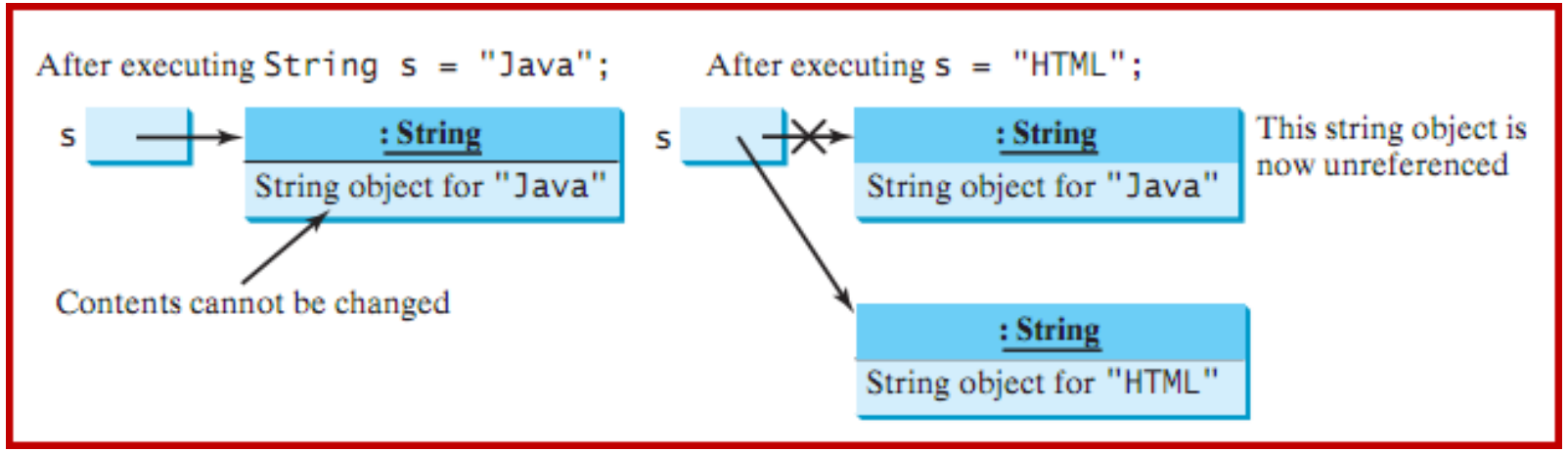
```
String str5 = new String("Hello").intern();
String str6 = new String("Hello").intern();
System.out.println(str5 == str6); // Output: true
```

Section 2

String Immutable

Immutable String and Interned String

- **String objects are immutable:**
  - String s = "Java"; //creates a String object
  - s = "HTML"; // creates a new String
- Original "Java" object still exists but is inaccessible since variable ‘s’ now points to the new object



Immutable String and Interned String

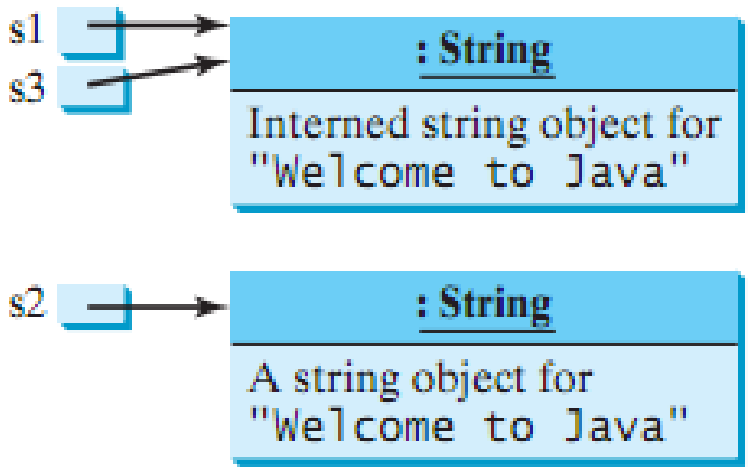
- Since strings immutable and ubiquitous JVM uses unique instance for same string literals (interning)
- Improves efficiency and saves memory:

```
String s1 = "Welcome to Java";

String s2 = new String("Welcome to Java");

String s3 = "Welcome to Java";

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



String Comparison

- We can compare string in Java on the basis of content and reference.
- It is used in **authentication** (by equals() method), **sorting** (by compareTo() method), **reference matching** (by == operator) etc.
- There are three ways to compare string in java:
  - ✓ In **authentication:** by equals()/equalsIgnoreCase() method
  - ✓ In **sorting:** by == operator
  - ✓ **Reference matching**: by compareTo() method. This method compares values **lexicographically** (từ vựng) and returns an integer value that describes if first string is *less than*, *equal to* or *greater* than second string.

String Comparisons

- How do you compare the contents of two strings?

```
if (s1 == s2)
    System.out.println("s1 and s2 are the same object");
else
    System.out.println("s1 and s2 are different objects");
```

- The == operator checks if two strings are same object reference
- The == operator does not compare string contents
- Use **equals**() to compare contents

```
if (s1.equal(s2))
    System.out.println("s1 and s2 are the same content");
else
    System.out.println(“s1 and s2 are different content");
```

Immutable String

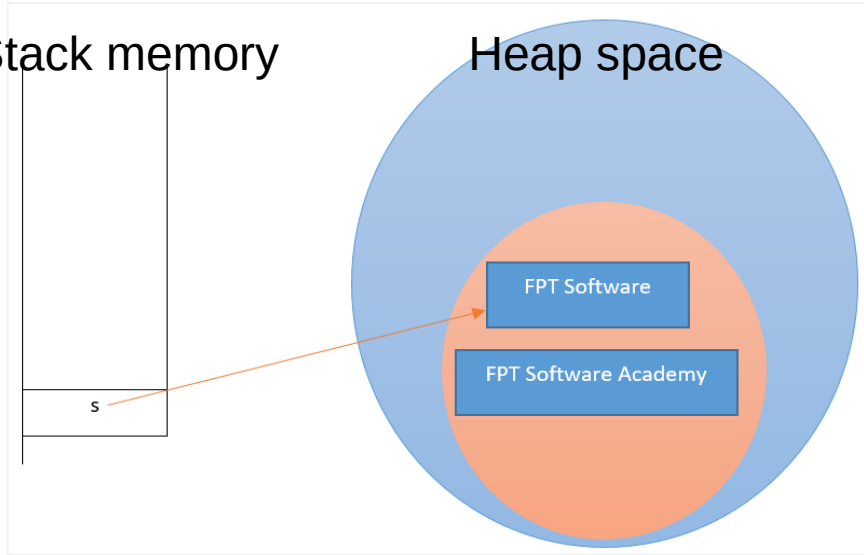
- **Example**:

```
public class TestImmutableString {

  public static void main(String[] args) {
    String s = "FPT Software";
    s.concat(" Academy"); // concat() method appends the string at the end
    System.out.println(s);// will print FPT Software because strings are
                          // immutable objects
  }
}
```

- **Output**:

```
FPT Software
```



Solution:

```
s = s.concat(" Academy");
```

Immutable String and Interned String

- **Substring extraction with no modification:**

```
String sentence = "The quick brown fox jumps over the lazy dog.";
String substring = sentence.substring(4, 10); // Creates a new string "brown fox"
System.out.println(sentence); // Still contains the complete sentence
System.out.println(substring); // Outputs "brown fox"
```

- **Methods that return new strings:**

✓ Many string methods operate on the original string but return a new string with the transformed content. Examples include `toLowerCase()`, `toUpperCase()`, `trim()`, etc.

```
String sentence = "The quick brown fox jumps over the lazy dog.";
String newSentence = sentence.toUpperCase();
System.out.println(sentence == newSentence); // Print 'false'
```

Section 3

String Methods

String class

- **Methods:**
  - ✓ **char charAt(int index)**: It returns the character at the specified index. Specified index value should be between 0 to length() -1 both inclusive. It throws IndexOutOfBoundsException if index<0||>= length of String.
  - ✓ **boolean equals(Object obj)**: Compares the string with the specified string and returns true if both matches else false.
  - ✓ **int compareTo(String string)**: This method compares the two strings based on the Unicode value of each character in the strings.
  - ✓ **boolean startsWith(String prefix)**: It tests whether the string is having specified prefix, if yes then it returns true else false.
  - ✓ **boolean endsWith(String suffix)**: Checks whether the string ends with the specified suffix.
  - ✓ **int hashCode()**: It returns the hash code of the string.
  - ✓ **int indexOf(int ch)**: Returns the index of first occurrence of the specified character ch in the string.
  - ✓ **boolean contains(CharSequence s)**: It checks whether the string contains the specified sequence of char values. If yes then it returns true else false. It throws NullPointerException of ‘s’ is null.

String class

- **Methods:**
  - ✓ **String concat(String str)**: Concatenates the specified string “str” at the end of the string.
  - ✓ **String trim()**: Returns the substring after omitting leading and trailing white spaces from the original string.
  - ✓ **byte[] getBytes()**: This method is similar to the above method it just uses the default charset encoding for converting the string into sequence of bytes.
  - ✓ **int length()**: It returns the length of a String.
  - ✓ **boolean matches(String regex)**: It checks whether the String is matching with the specified **regular expression** regex.
  - ✓ **static String valueOf()**: This method returns a string representation of passed arguments such as int, long, float, double, char and char array.
  - ✓ **char[] toCharArray()**: Converts the string to a character array.
  - ✓ **String[] split(String regex)**: Same as split(String regex, int limit) method however it does not have any threshold limit.

compareTo() method

- **compareTo** compares Strings: s1.**compareTo**(s2)
  - ✓0 if equal
  - ✓< 0 if s1 < s2
  - ✓> 0 if s1 > s2

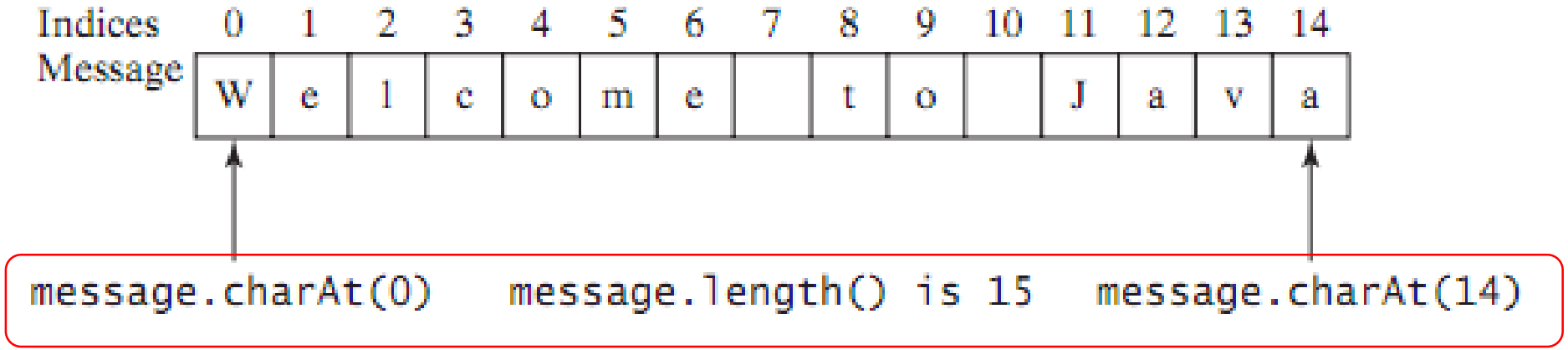
```
// Comparing strings lexicographically
String str1 = "apple";
String str2 = "banana";

// Compare str1 and str2
int result1 = str1.compareTo(str2);
System.out.println("Comparison of \"" + str1 + "\" and \"" + str2 + "\": " + result1);
//Output: Comparison of "apple" and "banana": -1
```

- **equalsIgnoreCase()**, **compareToIgnoreCase()**: ignore case when comparing

String Length, Characters, and Combining Strings

java.lang.String	
+length(): int	Returns the number of characters in this string.
+charAt(index: int): char	Returns the character at the specified index from this string.
+concat(s1: String): String	Returns a new string that concatenates this string with string s1.

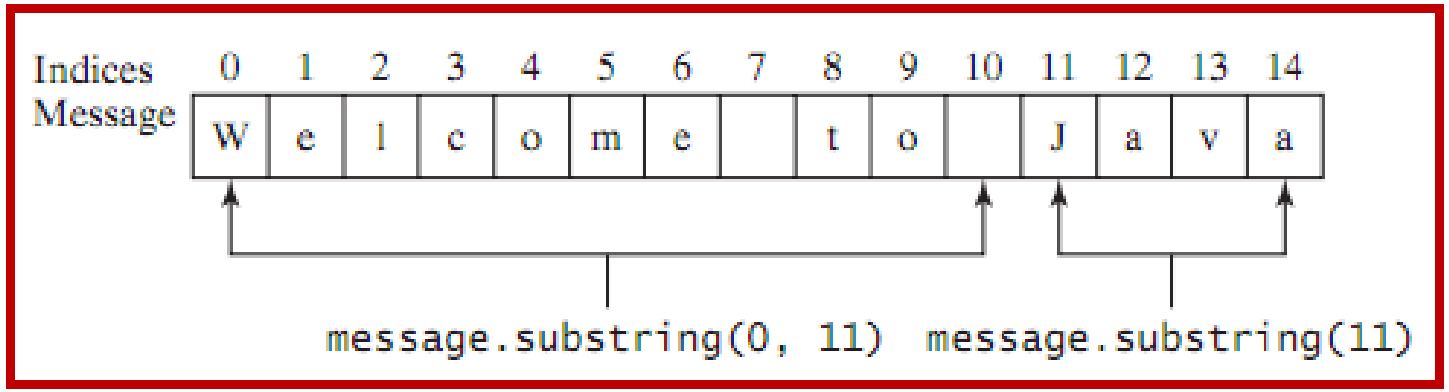


Obtaining Substrings

java.lang.String

```
+substring(beginIndex: int):
 String

+substring(beginIndex: int,
 endIndex: int): String
```



Converting, Replacing, and Splitting Strings

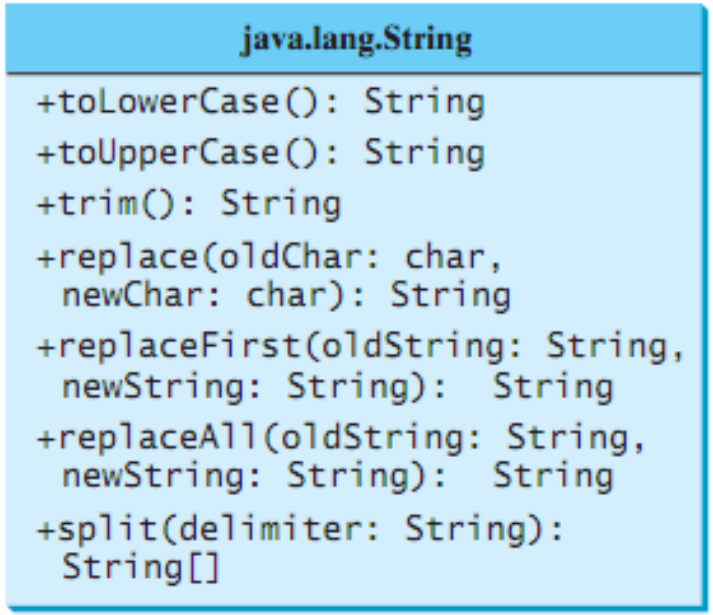
- **Example:**

```
String str = "Welcome";
String newStr = str.toLowerCase();
System.out.println(newStr);

newStr = str.toUpperCase();
System.out.println(newStr);

newStr = str.replaceFirst("e", "X"); //Output: WXlcome
System.out.println(newStr);

newStr = str.replace('e', 'X');
System.out.println(newStr); //Output: WXlcomX
```



- **split()** extracts tokens from a string based on delimiters

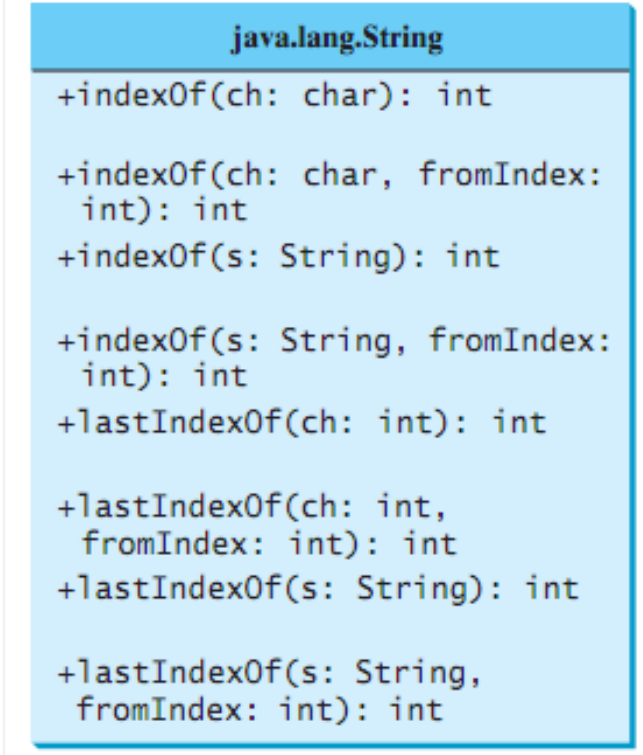
```
String[] tokens = "Java#HTML#Perl".split("#");
for (int i = 0; i < tokens.length; i++)
    System.out.print(tokens[i] + " ");
```

Finding a Character or a Substring in a String

- **indexOf()** and **lastIndexOf()** find chars/substrings in a string

```
"Welcome to Java".indexOf('W') returns 0.
"Welcome to Java".indexOf('o') returns 4.
"Welcome to Java".indexOf('o', 5) returns 9.
"Welcome to Java".indexOf("come") returns 3.
"Welcome to Java".indexOf("Java", 5) returns 11.
"Welcome to Java".indexOf("java", 5) returns -1.

"Welcome to Java".lastIndexOf('W') returns 0.
"Welcome to Java".lastIndexOf('o') returns 9.
"Welcome to Java".lastIndexOf('o', 5) returns 4.
"Welcome to Java".lastIndexOf("come") returns 3.
"Welcome to Java".lastIndexOf("Java", 5) returns -1.
"Welcome to Java".lastIndexOf("Java") returns 11.
```



Conversion between Strings and Arrays

- **Strings convert to/from char[] arrays:**
  - ✓String to char[]: `char[] chars = s.toCharArray();`
  - ✓char[] to String:

```
String s = new String(charArray);
String s = String.valueOf(charArray);
```

- **Example:**

```
String str = new String(new char[]{'J', 'a', 'v', 'a'});
String str = String.valueOf(new char[]{'J', 'a', 'v', 'a'});
```

Formatting Strings

- String.format creates formatted strings:

```
String.format(format, args...)
```

- **Example:**

```
String s = String.format("%5.2f", 45.556);
```

- Like printf but returns string rather than printing

Finger Exercise

- A string is a palindrome if it reads the same forward and backward. The words “mom,” “dad,” and “noon,” for instance, are all palindromes.
- Write a program that prompts the user to enter a string and reports whether the string is a palindrome.

```
public static boolean isPalindrome(String s) {
    // The index of the first character in the string
    int low = 0;
    // The index of the last character in the string
    int high = s.length() - 1;
    while (low < high) {
        if (s.charAt(low) != s.charAt(high))
            return false; // Not a palindrome
        low++;
        high--;
    }
    return true; // The string is a palindrome
}
```

String/Number casting

- **Convert a digit sequence to number**

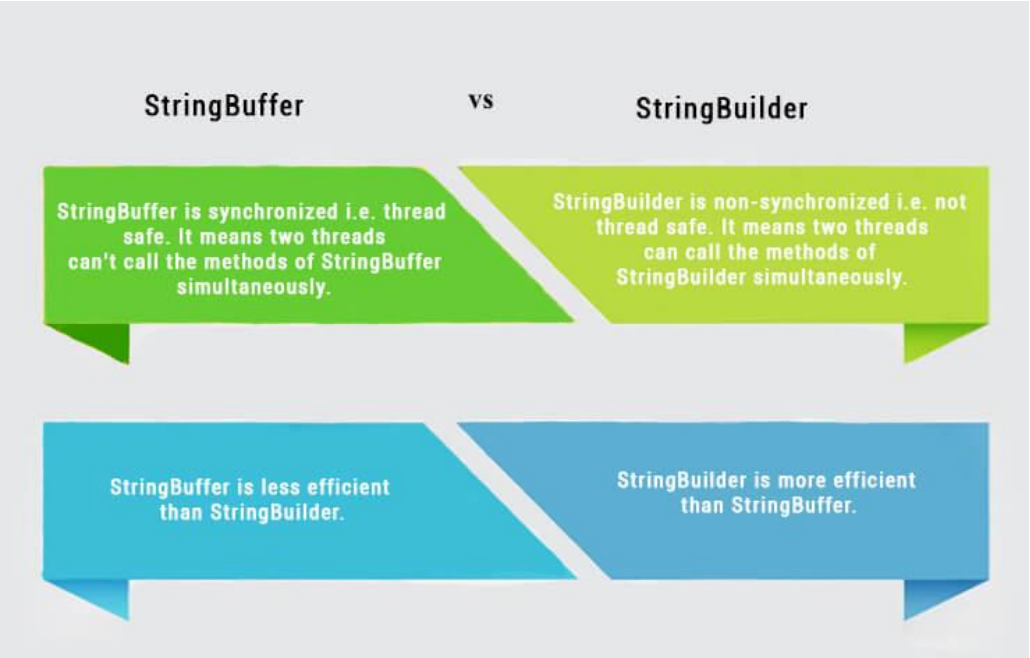
```
// Each class in right hand side is called wrapper
// class of the corresponding primitive type
byte  b = Byte.parseByte("128"); // NumberFormatException
short s = Short.parseShort("32767");
int   x = Integer.parseInt("2");
int   y = Integer.parseInt("2.5");// NumberFormatException
int   z = Integer.parseInt("a"); // NumberFormatException
long  l = Long.parseLong("15");
float f = Float.parseFloat("1.1");
double d = Double.parseDouble("2.5");
```

Section 4

StringBuilder and StringBuffer

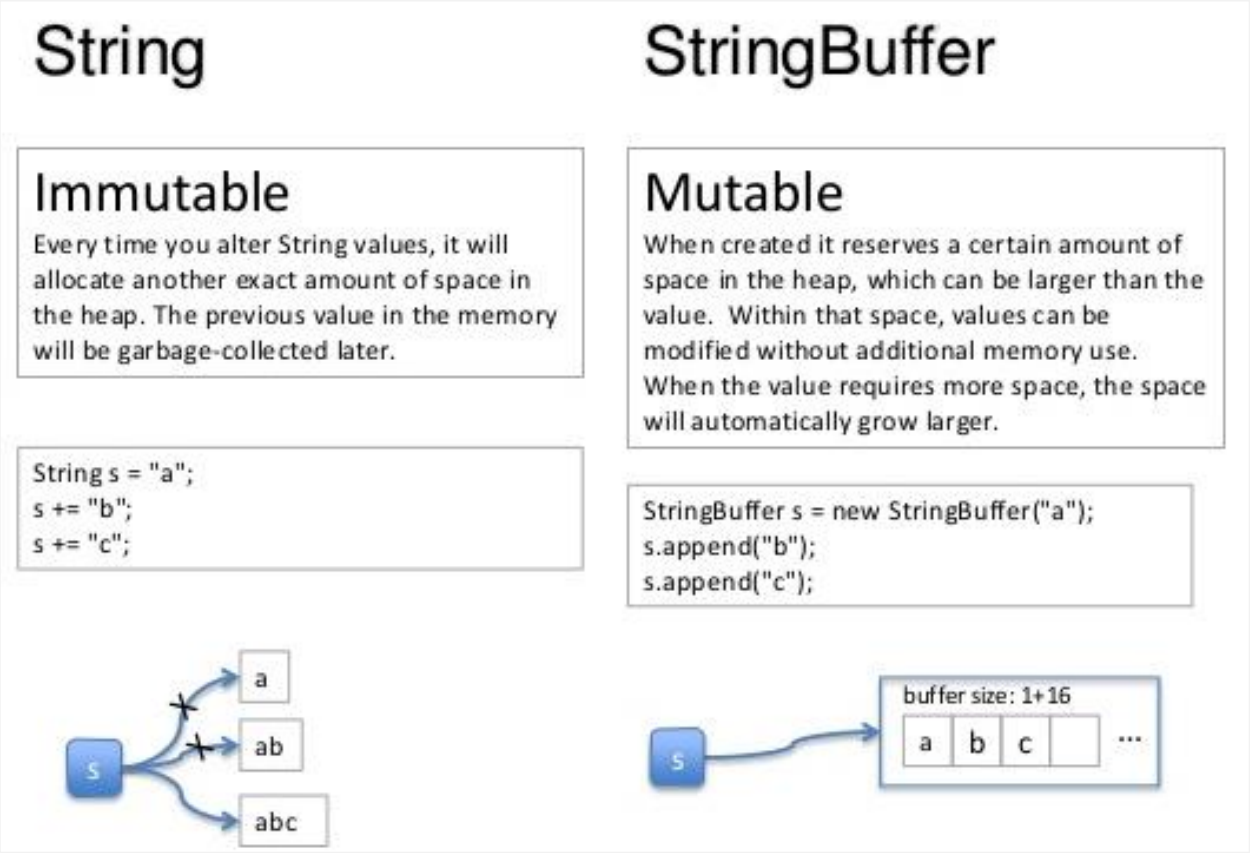
StringBuilder and StringBuffer classes

- **StringBuilder** / **StringBuffer** alternate String classes
  - ✓ Allows modifying contents unlike immutable String
  - ✓ Can add, insert, append to StringBuilder/Buffer
  - ✓ **StringBuffer** is synchronized, thread-safe
  - ✓ **StringBuilder** faster for single thread use



String/StringBuilder/StringBuffer

- String is **immutable**, if you try to alter their values, another object gets created,
- StringBuffer and StringBuilder are **mutable** so they can change their values.



Methods of StringBuffer class

Method	Description
public synchronized StringBuffer **append**(String s)	Is used to append the specified string with this string. The append() method is overloaded like: ✓ append(char), ✓ append(boolean), ✓ append(int), ✓ append(float), ✓ append(double) etc.
public synchronized StringBuffer **insert**(int offset, String s)	Is used to insert the specified string with this string at the specified position. The insert() method is overloaded like: ✓ insert(int, char), insert(int, boolean), ✓ insert(int, int), insert(int, float), insert(int, double) etc.
public synchronized StringBuffer **replace**(int startIndex, int endIndex, String str)	Is used to replace the string from specified startIndex and endIndex.
public synchronized StringBuffer **delete**(int startIndex, int endIndex)	Is used to delete the string from specified startIndex and endIndex.
public synchronized StringBuffer reverse()	Is used to reverse the string.

Methods of StringBuffer class

Method	Description
public int **capacity**()	Is used to return the current capacity.
public void **ensureCapacity**(int minimumCapacity)	Is used to ensure the capacity at least equal to the given minimum.
public char **charAt**(int index)	Is used to return the character at the specified position.
public int **length**()	Is used to return the length of the string i.e. total number of characters.
public String **substring**(int beginIndex)	Is used to return the substring from the specified **beginIndex**.
public String **substring**(int beginIndex, int endIndex)	Is used to return the substring from the specified **beginIndex** and **endIndex**.

StringBuilder class

java.lang.StringBuilder	
+append(data: char[]): StringBuilder	Appends a char array into this string builder.
+append(data: char[], offset: int, len: int): StringBuilder	Appends a subarray in data into this string builder.
+append(v: *aPrimitiveType*): StringBuilder	Appends a primitive type value as a string to this builder.
+append(s: String): StringBuilder	Appends a string to this string builder.
+delete(startIndex: int, endIndex: int): StringBuilder	Deletes characters from startIndex to endIndex-1.
+deleteCharAt(index: int): StringBuilder	Deletes a character at the specified index.
+insert(index: int, data: char[], offset: int, len: int): StringBuilder	Inserts a subarray of the data in the array to the builder at the specified index.
+insert(offset: int, data: char[]): StringBuilder	Inserts data into this builder at the position offset.
+insert(offset: int, b: *aPrimitiveType*): StringBuilder	Inserts a value converted to a string into this builder.
+insert(offset: int, s: String): StringBuilder	Inserts a string into this builder at the position offset.
+replace(startIndex: int, endIndex: int, s: String): StringBuilder	Replaces the characters in this builder from startIndex to endIndex-1 with the specified string.
+reverse(): StringBuilder	Reverses the characters in the builder.
+setCharAt(index: int, ch: char): void	Sets a new character at the specified index in this builder.

Modifying Strings in the StringBuilder

- Can **append**, **insert**, **delete**, **replace** on StringBuilder
- Overloaded **append** methods for various types like boolean, char, double, String etc.

```
StringBuilder stringBuilder = new StringBuilder();
stringBuilder.append("Welcome");
stringBuilder.append(' ');
stringBuilder.append("to");
stringBuilder.append(' ');
stringBuilder.append("Java");
//Output: Welcome to Java
stringBuilder.insert(11, "HTML and ");
//Output: Welcome to HTML and Java
```

```
StringBuilder sb = new StringBuilder("Hello World");
int offset = 6;
int number = 123;

sb.insert(offset, number);

System.out.println(sb.toString());
```

Modifying Strings in the StringBuilder

- Delete characters from a string in the builder using the two delete methods
- Reverse the string using the reverse method
- Replace characters using the replace method,
- Set a new character in a string using the setCharAt method.

```
//stringBuilder = "Welcome to Java";
//1. Changes the builder to Welcome Java.
stringBuilder.delete(8, 11);            //2. Changes the builder to Welcome o Java.
stringBuilder.deleteCharAt(8);          //3. Changes the builder to avaJ ot emocleW.
stringBuilder.reverse();                //4. Changes the builder to Welcome to HTML
stringBuilder.replace(11, 15, "HTML");//5. Sets the builder to welcome to Java.
stringBuilder.setCharAt(0, 'w');
```

StringBuilder and StringBuffer class

- The Java **StringBuilder** class is same as **StringBuffer** class except that it is non-synchronized.

No.	StringBuffer	StringBuilder
1)	StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.

- **Example**:

```
public class ConcatTest {
  public static void main(String[] args) {
    long startTime = System.currentTimeMillis();

    StringBuffer sb = new StringBuffer("Java");
    for (int i = 0; i < 1000000; i++) {
      sb.append(" Learning");
    }

    System.out.println("Time taken by StringBuffer: "
        + (System.currentTimeMillis() - startTime) + "ms");

    startTime = System.currentTimeMillis();

    StringBuilder sb2 = new StringBuilder("Java");
    for (int i = 0; i < 1000000; i++) {
      sb2.append(" Learning");
    }

    System.out.println("Time taken by StringBuilder: "
        + (System.currentTimeMillis() - startTime) + "ms");
  }
}
```

- **Output:**

```
Time taken by StringBuffer: 51ms
Time taken by StringBuilder: 26ms
```

Using StringTokenizer Class

- **`StringTokenizer`** can be used to parse a line into words
  - ✓import `java.util.*`
  - ✓some of its useful methods are shown in the text
    - e.g. test if there are more tokens
  - ✓you can specify *delimiters* (the character or characters that separate words)
    - the default delimiters are "white space" (space, tab, and newline)

Example: `StringTokenizer`

- Display the words separated by any of the following characters: space, new line (\n), period (.) or comma (,).

```
String inputLine = keyboard.nextLine();
StringTokenizer wordFinder = new StringTokenizer(inputLine, " \n.,");
//the second argument is a string of the 4delimiters
while(wordFinder.hasMoreTokens()){
    System.out.println(wordFinder.nextToken());
}
```

Entering "`Question,2b.or !tooBee.`" gives this output:

```
Question
2b
or
!tooBee
```

Section 3

Regular Expression

Regular Expression

- Regular expression (regex) describes a pattern to match strings
- Can *match*, *replace*, *split* strings using a regex pattern
- Very useful and powerful

Matching, Replacing, Splitting by Patterns

- **matches**() method like **equals**(), e.g.:

```
"Java".matches("Java");
"Java".equals("Java");
```

- **matches**() can match patterns, not just fixed strings
  - ✓ *"Java.*" is a regex describing the pattern: Begins with Java followed by any characters*

```
"Java is fun".matches("Java.*");
"Java is cool".equals("Java.*");
"Java is powerful".matches("Java.*");
```

- Matching a Date in "**YYYY-MM-DD**" Format
  - ✓ *\\d matches a digit*

```
String date = "2023-11-29";
String regexDate = "^\\d{4}-\\d{2}-\\d{2}$";
boolean isMatch = date.matches(regexDate);
```

Matching, Replacing, Splitting by Patterns

- **replaceAll()**, **replaceFirst()**, **split()** methods can be used with regexes

✓ **Replace all digits with "X":**

```
String text = "The price is $19.99 and the quantity is 25.";
String replacedText = text.replaceAll("\\d", "X");
System.out.println(replacedText);          //Output: The price is $XX.XX and the quantity is XX.
```

✓ **Replaces $, +, # with NNN:**

```
String s = "a+b$#c".replaceAll("[$+#]", "NNN");
System.out.println(s); //Output: aNNNbNNNNNNc
```

✓ **Replace the first word:**

```
String sentence = "Java11 is a powerful programming language."; // Replace the first word with "Python"
String replacedText2 = sentence.replaceFirst("\\b\\w+\\b", "Python");
System.out.println("Sentence after replacement: " + replacedText2);
```

\\b\\w+\\b matches a word boundary (\\b), one or more word characters (\\w+), and another word boundary.

Matching, Replacing, Splitting by Patterns

- **Can split on regex delimiters:**

```
String[] tokens = "Java,C?C#,C++".split("[.,:;?]");
for (int i = 0; i < tokens.length; i++) {
    System.out.println(tokens[i]);
}
```

✓ *[.,:;?] matches the punctuation marks*

✓ *Splits into Java, C, C#, and C++*

- **Split a string on punctuation:**

```
String text2 = "Java is a powerful, versatile programming language; it is used widely.";
// Split the text on punctuation
String[] segments = text2.split("\\p{Punct}+");
System.out.println(Arrays.toString(segments));
//Output: [Java is a powerful,  versatile programming language,  it is used widely]
```

✓ *The regular expression \\p{Punct}+ matches one or more punctuation characters.*

Summary

01. Creating Strings

02. String Immutable

03. String Methods

04. StringBuilder and StringBuffer

05. Regular Expression

References

- *https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html*
- *https://docs.oracle.com/javase/tutorial/java/data/strings.html*
- *https://www.javatpoint.com/java-string*

Questions

THANK YOU!