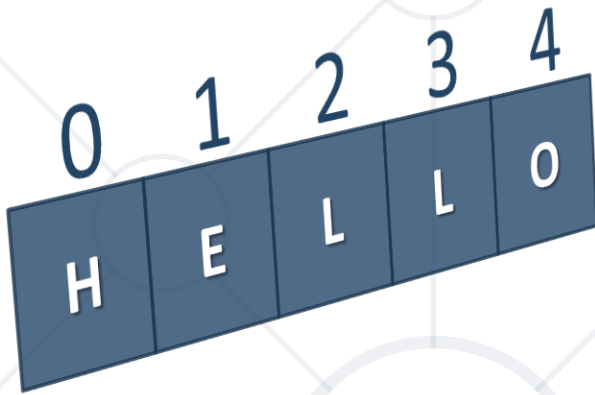


Text Processing

Manipulating Text



SoftUni Team
Technical Trainers



SoftUni



Software University

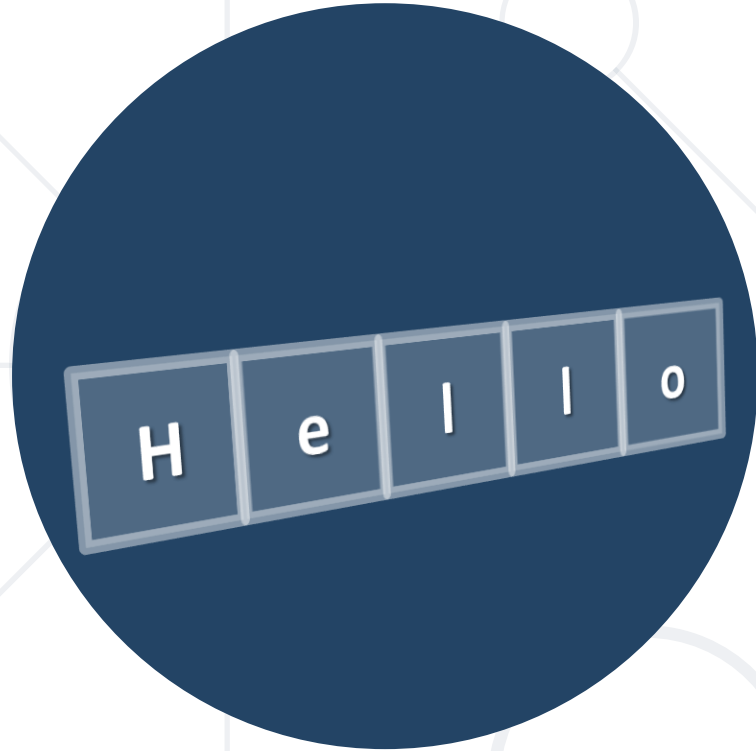
<https://softuni.bg>

sli.do

#fund-java

1. What Is a **String**?
2. **Manipulating** Strings
3. **Building** and **Modifying** Strings
 - Using **StringBuilder** Class
 - Why **Concatenation** Is a Slow Operation?






What is a String?

Strings

What is a String?

- 
- Strings are **sequences** of characters (texts)
 - The string data type in Java
 - Declared by the **String**
 - Strings are enclosed in double quotes:

```
String text = "Hello, Java";
```

Strings Are Immutable

- Strings are **immutable** (read-only) sequences of characters
- Accessible by **index** (read-only)

```
String str = "Hello, Java";  
char ch = str.charAt(2); // l
```

- Strings use **Unicode** (can use most alphabets, e.g. Arabic)

```
String greeting = "你好"; // (Lí-hó) Taiwanese
```



Initializing a String

- Initializing from a string literal:

```
String str = "Hello, Java";
```

- Reading a **string** from the console:

```
String name = sc.nextLine();  
System.out.println("Hi, " + name);
```

- Converting a **string** from and to a **char array**:

```
String str = new String(new char[] {'s', 't', 'r'});  
char[] charArr = str.toCharArray();  
// ['s', 't', 'r']
```





Manipulating Strings

Concatenating

- Use the **+** or the **+=** operators

```
String text = "Hello" + ", " + "world!";  
// "Hello, world!"
```

```
String text = "Hello, ";  
text += "John"; // "Hello, John"
```

- Use the **concat()** method

```
String greet = "Hello, ";  
String name = "John";  
String result = greet.concat(name);  
System.out.println(result); // "Hello, John"
```



Joining Strings

- `String.join("", ...)` concatenates strings

```
String t = String.join("", "con", "ca", "ten", "ate");  
// "concatenate"
```

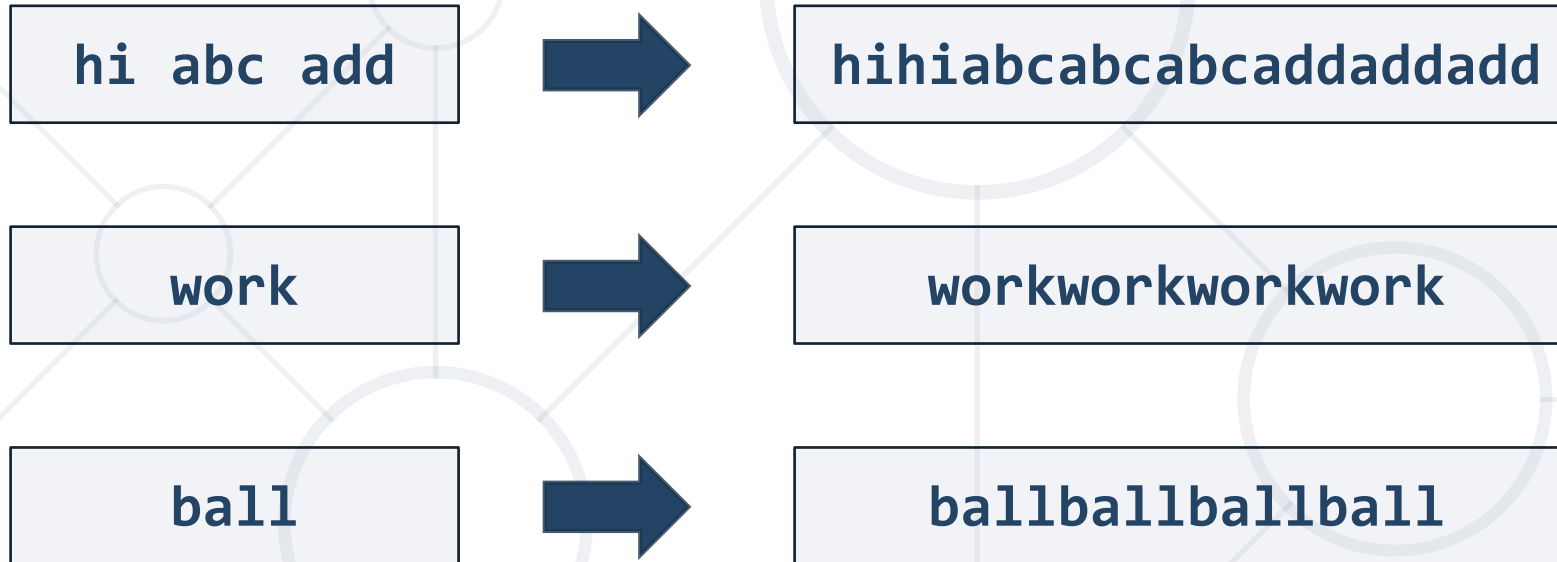
- Or an array/list of strings
 - Useful for repeating a string

```
String s = "abc";  
String[] arr = new String[3];  
for (int i = 0; i < arr.length; i++) { arr[i] = s; }  
String repeated = String.join("", arr); // "abcabcabc"
```



Problem: Repeat Strings

- Read an array from strings
- Repeat each word **n** times, where **n** is the length of the word



Check your solution here: <https://alpha.judge.softuni.org/contests/text-processing-lab/1669>

Solution: Repeat Strings

```
String[] words = sc.nextLine().split(" ");  
List<String> result = new ArrayList<>();  
for (String word : words) {  
    result.add(repeat(word, word.length()));  
}  
System.out.println(String.join("", result));
```

Solution: Repeat Strings

```
static String repeat(String s, int repeatCount) {  
    String[] repeatArr = new String[repeatCount];  
    for (int i = 0; i < repeatCount; i++) {  
        repeatArr[i] = s;  
    }  
    return String.join("", repeatArr);  
}
```

Substring

- **substring(int startIndex, int endIndex)**

```
String card = "10C";  
String power = card.substring(0, 2);  
System.out.println(power); // 10
```

- **substring(int startIndex)**

```
String text = "My name is John";  
String extractWord = text.substring(11);  
System.out.println(extractWord); // John
```



Searching

- **indexOf()** - returns the first match index or -1

```
String fruits = "banana, apple, kiwi, banana, apple";  
System.out.println(fruits.indexOf("banana"));    // 0  
System.out.println(fruits.indexOf("orange"));    // -1
```

- **lastIndexOf()** - finds the last occurrence

```
String fruits = "banana, apple, kiwi, banana, apple";  
System.out.println(fruits.lastIndexOf("banana")); // 21  
System.out.println(fruits.lastIndexOf("orange")); // -1
```



Searching

- **contains()** - checks whether one string contains another

```
String text = "I love fruits.";
System.out.println(text.contains("fruits"));
// true
System.out.println(text.contains("banana"));
// false
```



Problem: Substring

- You are given a **remove word** and a **text**
- Remove all substrings that are equal to the remove word

ice
kicegiceiceb



kgb

key
keytextkey



text

abc
tabctqw



ttqw

word
wordawordbwordc



abc

Check your solution here: <https://alpha.judge.softuni.org/contests/text-processing-lab/1669>

Solution: Substring

```
String key = sc.nextLine();
String text = sc.nextLine();

int index = text.indexOf(key);
while (index != -1) {
    text = text.replace(key, "");
    index = text.indexOf(key);
}

System.out.println(text);
```

Splitting

- **Split** a string by a given **pattern**

```
String text = "Hello, john@softuni.bg, you have been  
using john@softuni.bg in your registration";
```

```
String[] words = text.split(", ");
```

```
// words[]: "Hello", "john@softuni.bg", "you have been..."
```

- **Split** by **multiple separators**

```
String text = "Hello, I am John.";
```

```
String[] words = text.split("[, .]+");
```

```
// "Hello", "I", "am", "John"
```



Replacing

- `replace(match, replacement)` - replaces **all** occurrences
 - The result is a new **string** (strings are **immutable**)



```
String text = "Hello, john@softuni.bg, you have been  
using john@softuni.bg in your registration.";  
String replacedText = text  
    .replace("john@softuni.bg", "john@softuni.com");  
System.out.println(replacedText);  
// Hello, john@softuni.com, you have been using  
john@softuni.com in your registration.
```

Problem: Text Filter

- You are given a string of banned words and a text
 - Replace all banned words in the text with asterisks (*)

Linux, Windows

It is not Linux, it is GNU/Linux. Linux is merely the kernel, while GNU adds the functionality...



It is not *****, it is GNU/*****. ***** is merely the kernel, while GNU adds the functionality...

Solution: Text Filter

```
String[] banWords = sc.nextLine.split(", ");
String text = sc.nextLine();
for (String banWord : banWords) {
    if (text.contains(banWord)) {
        String replacement = repeatStr("*",
            banWord.length());
        text = text.replace(banWord, replacement);
    }
}
System.out.println(text);
```

contains(...) checks if
string contains
another string

replace() a word with a sequence
of asterisks of the same length

```
private static String repeatStr(String str, int length) {  
    String replacement = "";  
    for (int i = 0; i < length; i++) {  
        replacement += str;  
    }  
    return replacement;  
}
```



Building and Modifying Strings

Using the StringBuilder Class

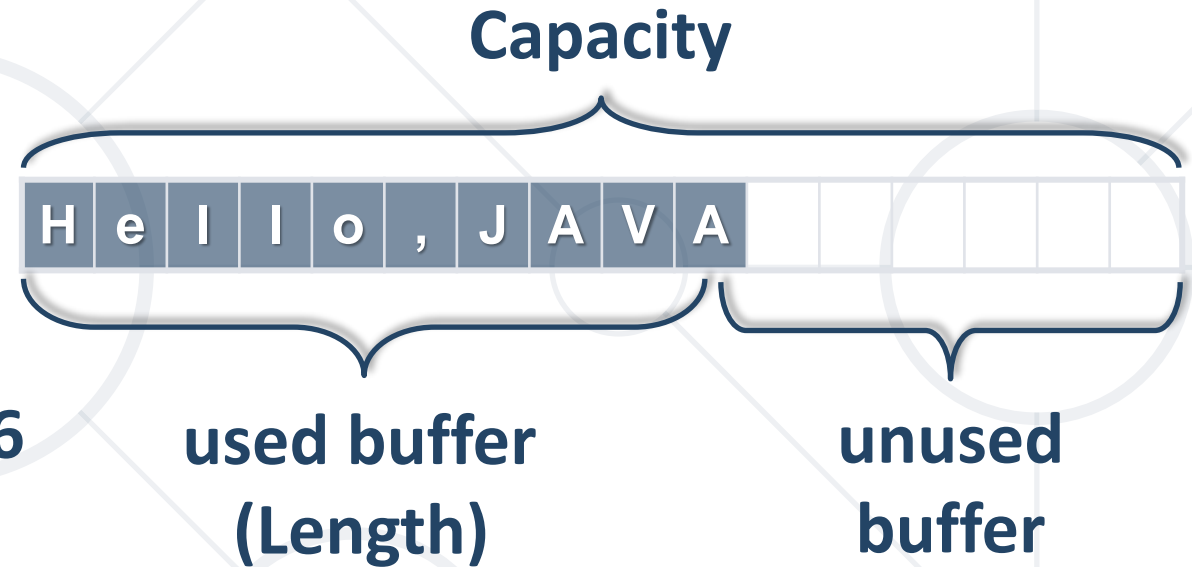
StringBuilder: How It Works?



StringBuilder:

`length() = 10`

`capacity() = 16`



- **StringBuilder** keeps a buffer space, allocated in advance
 - Do not allocate memory for most operations → performance

Using StringBuilder Class

- Use the **StringBuilder** to build/modify strings

```
StringBuilder sb = new StringBuilder();  
sb.append("Hello, ");  
sb.append("John! ");  
sb.append("I sent you an email.");  
System.out.println(sb.toString());  
// Hello, John! I sent you an email.
```



Concatenation vs. StringBuilder

- **Concatenating** strings is a **slow** operation because each iteration **creates** a **new string**

```
System.out.println(new Date());  
String text = "";  
for (int i = 0; i < 1000000; i++)  
    text += "a";  
System.out.println(new Date());
```



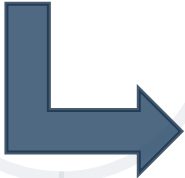

```
Tue Jul 10 13:57:20 EEST 2018  
Tue Jul 10 13:58:07 EEST 2018
```



Concatenation vs. StringBuilder

- Using **StringBuilder**

```
System.out.println(new Date());  
StringBuilder text = new  
StringBuilder();  
for (int i = 0; i < 1000000; i++)  
    text.append("a");  
System.out.println(new Date());
```



```
Tue Jul 10 14:51:31 EEST 2018  
Tue Jul 10 14:51:31 EEST 2018
```



StringBuilder Methods

- **append()** - appends the string representation of the argument

```
StringBuilder sb = new StringBuilder();  
sb.append("Hello Peter, how are you?");
```

- **length()** - holds the length of the string in the buffer

```
sb.append("Hello Peter, how are you?");  
System.out.println(sb.length()); // 25
```

- **setLength(0)** - removes all characters



StringBuilder Methods

- **charAt(int index)** - returns char on index

```
StringBuilder sb = new StringBuilder();  
sb.append("Hello Peter, how are you?");  
System.out.println(sb.charAt(1)); // e
```

- **insert(int index, String str)** –
inserts a string at the specified character position

```
sb.insert(11, " Ivanov");  
System.out.println(sb);  
// Hello Peter Ivanov, how are you?
```



StringBuilder Methods

- **replace(int startIndex, int endIndex, String str)** - replaces the chars in a substring

```
sb.append("Hello Peter, how are you?");  
sb.replace(6, 11, "George");
```

- **toString()** - converts the value of this instance to a String

```
String text = sb.toString();  
System.out.println(text);  
// Hello George, how are you?
```



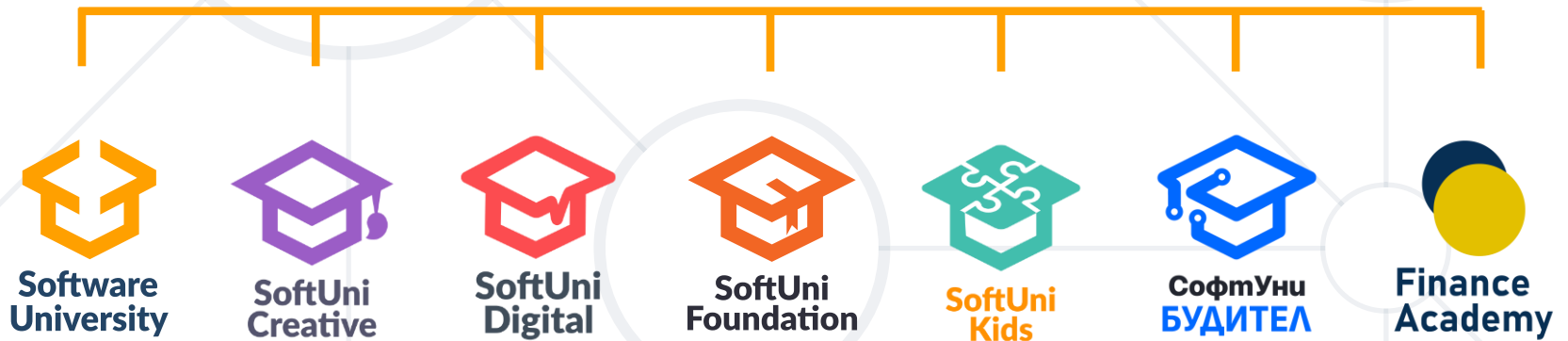
- Strings are **immutable** sequences of Unicode characters
- String processing methods
 - **concat(), indexOf(), contains(), substring(), split(), replace(), ...**
- **StringBuilder** efficiently builds/modifies strings



Questions?



SoftUni



SoftUni Diamond Partners



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, about.softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>

