String Processing

Processing and Manipulating Strings with State Machines and Regex

SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Contents

1. Manipulating Strings
   - Comparing, Concatenating, Searching
   - Extracting, Replacing, Deleting Substrings
   - Using the **StringBuilder** Class
2. State Machines
   - For String Processing
3. Regular Expressions

Have a Question?

sli.do

#JavaAdvanced

Formatting Strings

Basic Format Options

Formatting Strings

- `String.format()` supports common **numeric**, **String**, and **date/time** formats and **alignments**

- Inserting Strings

First argument

Second argument

```
String result = String.format("%1$s, %2$s!", "Hello", "World");
// Hello, World!
result = String.format("%2$s, %1$s!", "Hello", "World");
// World, Hello!
```

Initial symbol

Argument order

Separator

String format specifier

Formatting Strings (2)

- Inserting numbers

Integer format specifier

```
int number1 = 10;
String value = String.format(“Integer: %1$d", number1);       // 10
double number2 = 1.23456;
String value = String.format("Precision 3: %1$.3f", number2); // 1.234
```

Number precision

Double format specifier

- Padding

```
String value = String.format("%1$-10s || %2$10s", 1.26, 5.55);
System.out.println(value);
//1.26       ||       5.55
```

Right padding

Left padding

Problem: Student's Results

- Read a student's **name** and **results** for his courses
- Print the results in **columns** with precision of **2**
- Calculate his **average** score with precision of **4**

```
Gosho - 3.33333, 4.4444, 5.555
```

Name	JAdv	JavaOOP	AdvOOP	Average
Gosho	3.33	4.44	5.56	4.4442

Check your solution here: https://judge.softuni.bg/Contests/777

Solution: Student's Results

```
//TODO: read student's name and results

System.out.println(
  String.format("%1$-10s|%2$7s|%3$7s|%4$7s|%5$7s|",
 "Name", "JAdv", "JavaOOP", "AdvOOP", "Average"));

double average =
  (results.get(0) + results.get(1) + results.get(2)) / 3;

System.out.println(
String.format("%1$-10s|%2$7.2f|%3$7.2f|%4$7.2f|%5$7.4f|",
name, results.get(0), results.get(1), results.get(2),average));
```

Check your solution here: https://judge.softuni.bg/Contests/777

Manipulating Strings

Comparing, Concatenating, Searching, Extracting Substrings, Splitting

Trimming Whitespaces and boolean methods

- `str.trim()` – removes **all** whitespaces at start and end

```java
String s = "    example of white space    ";
String clean = s.trim();
// "example of white space"
```

- `str.startsWith(String prefix)`

```java
String s = "C# is the best!";
boolean startsWithJava = s.startsWith(“Java");
System.out.println(startsWithJava); // false
```

- `str.endsWith(String suffix)`

```java
String s = "How are you?";
boolean isQuestion = s.endsWith("?");
System.out.println(isQuestion); // true
```

Searching in Strings

- Finding a character or substring within a given String
  - **`str.indexOf(String/char term)`** – returns the index of the **first** occurrence of **`term`** in **`str`**
    - Returns **`-1`** if there is no match

```
String email = "vasko@gmail.org";
int firstIndex = email.indexOf("@gmail.org"); // 5
int noIndex = email.indexOf('@', 6); // -1
```

  - **`str.lastIndexOf(String/char term)`** – returns the index of the **last** occurrence of **`term`** in **`str`**

```
String verse = "To be or not to be..";
int lastIndex = verse.lastIndexOf("be"); // 16
```

Extracting Substrings

- `str.substring(int startIndex, int endIndex)`

```
String filename = "C:\\Pics\\Rila2017.jpg";
String name = filename.substring(8, 16);
// name is Rila2017
```

- `str.substring(int startIndex)`

```
String filename = "C:\\Pics\\Rila2017.jpg";
String nameAndExtension = filename.substring(8);
// nameAndExtension is Summer2017.jpg
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
C	:	\	P	i	c	s	\	R	i	l	a	2	0	1	7	.	j	p	g

Splitting Strings

- To **split** a String by given **separator(s)** use the following method:
  - Single separator

```
String line = “Carrot:Orange,Apple:Red";
String[] vegetables = line.split(",");
```

  - Multiple separators

```
String line = “Carrot:Orange,Apple:Red";
String[] vegetables = line.split("[,:]");
```

```
Carrot
Orange
Apple
Red
```

Problem: Parse URL

- Write a program that parses an **URL** address given in the format:
  - **[protocol]://[server]/[resource]**
- Extract protocol, server and resource

```
https://softuni.bg/trainings/1531/java-advanced-january-2017
```

```
Protocol = https
Server = softuni.bg
Resources =
trainings/1531/java-
advanced-january-2017
```

Check your solution here: https://judge.softuni.bg/Contests/777

Solution: Parse URL

```
String[] reminder = input.split("://");
String protocol = reminder[0];

int serverEndIndex = reminder[1].indexOf("/");
String server = reminder[1].substring(0, serverEndIndex);
String resource = reminder[1].substring
          (serverEndIndex + 1, reminder[1].length() - 1);
```

Check your solution here: https://judge.softuni.bg/Contests/777

Changing Character Casing

- Using the method `toLowerCase()`

```
String alpha = "aBcDeFg";
String lowerAlpha = alpha.toLowerCase();
System.out.println(lowerAlpha);
// abcdefg
```

- Using the method `toUpperCase()`

```
String alpha = "aBcDeFg";
String upperAlpha = alpha.toUpperCase();
System.out.println(upperAlpha);
// ABCDEFG
```

Replacing substrings

- **`str.replace(CharSequence tar, CharSequence rep)`** – replaces all occurrences of a given **String** with another

```
String cocktail = "Vodka + Martini + Cherry";
String replaced = cocktail.replace("+", "and");
// Vodka and Martini and Cherry
```

- **`str.replaceFirst(String str, String rep)`** – replaces only the **first** match of a given **String** with another

```
String str = "My number is: +123123123";
String newStr = str.replaceFirst("12", "9");
// My number is: +93123123
```

Problem: Parse Tags

- Write a program that changes the text in all regions surrounded by the tags **<upcase>** and **</upcase>** to upper-case.
- The tags **cannot** be nested.

```
We are living in a <upcase>yellow submarine</upcase>. We don't have <upcase>anything </upcase> else.
```

```
We are living in a YELLOW SUBMARINE. We don't have ANYTHING else.
```

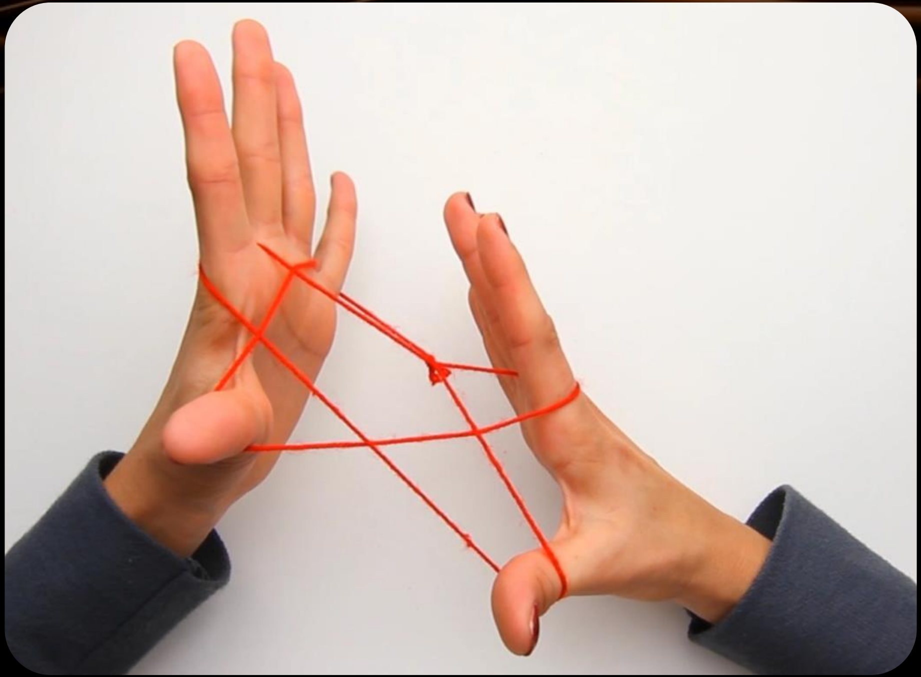
Check your solution here: https://judge.softuni.bg/Contests/777

Solution: Parse Tags

```java
//TODO: Read text
while (input.contains(upcaseStart)) {
  int startIndex = input.indexOf(upcaseStart);
  int endIndex = input.indexOf(upcaseStop);

  String reminder = input.substring(startIndex + 8);
  String upReminder = reminder.toUpperCase();
  input = input.replaceFirst(reminder, upReminder);
  input = input.replaceFirst(upcase, "");
  input = input.replaceFirst(upcaseStop, "");
}
//TODO: Write modified text
```

Check your solution here: https://judge.softuni.bg/Contests/777



Comparing Strings

Difference between == and .equals()

Comparing Strings

- Equality checking by operator **==**
  - **WARNING!** Compares **references**, not the content of the Strings

```
if (str1 == str2) {
    …
}
```

✗

- Using the **equals()** and **equalsIgnoreCase()** method
  - Unlike the operator **==** these methods compare Strings by their value

```
if (str1.equals(str2)) {
    …
}
```

✓

Comparing Strings (2)

- Dictionary-based String comparison
  - Case-sensitive

```
int result = str1.compareTo(str2);
```

  - Case-insensitive

```
int result = str1.compareToIgnoreCase(str2);
// result == 0 if str1 equals str2
// result < 0 if str1 is before str2
// result > 0 if str1 is after str2
```

Concatenating and Building Strings

Using the StringBuilder Class

Concatenating Strings

- There are two ways to **combine** Strings:
  - Using the **`concat()`** method

```
String str = str.concat(strToConcat);
```

  - Using the **+** or the **+=** operators

```
String str = str1 + str2 + str3;
String str += str1;
```

- Any object can be appended to a String

```
String name = "Peter";
int age = 22;
String s = name + " " + age; // → "Peter 22"
```

Changing the Contents of a String

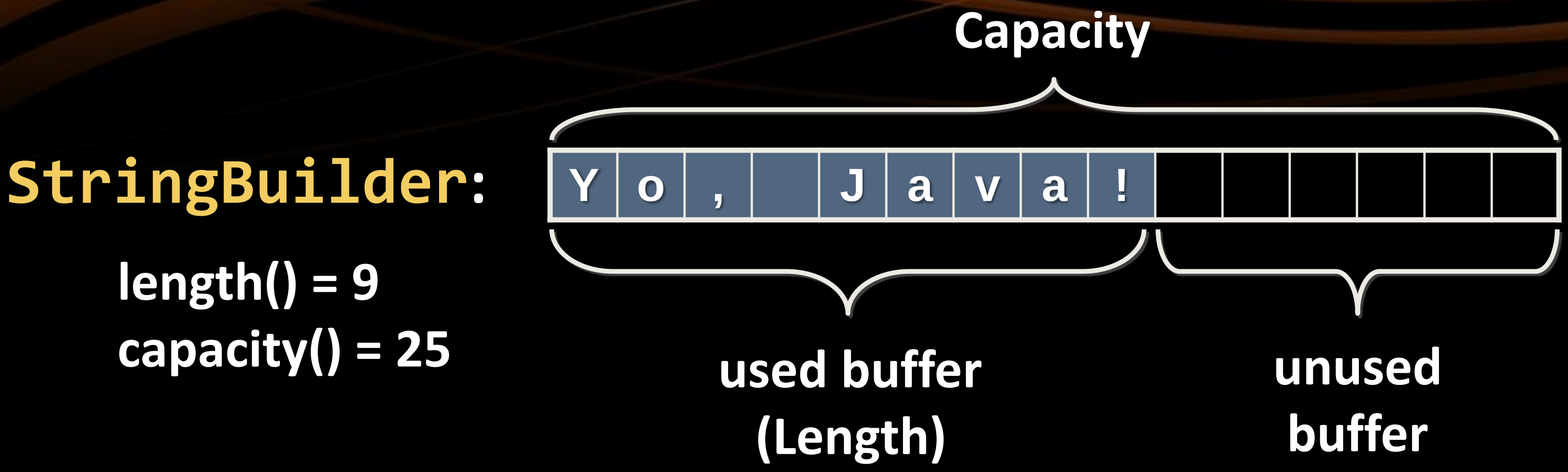
- Use the **`java.lang.StringBuilder`** class for modifiable Strings of characters:

```
public static String reverseString(String s) {
    StringBuilder sb = new StringBuilder();

    for (int i = s.length() - 1; i >= 0; i--) {
        sb.append(s.charAt(i));
    }
    return sb.toString();
}
```

  - Use **`StringBuilder`** if you need to keep adding characters to a String or when you have to print to the console many times

StringBuilder: How It Works?



- **StringBuilder** keeps a buffer memory, allocated in advance
  - Most operations use the **buffer memory** and do not allocate new objects
  - Using StringBuilder is **faster** than simple String concatenation

The StringBuilder Class

- `insert(int index, String str)` - inserts a String at a certain index

```
StringBuilder sb = new StringBuilder("123456");
sb.insert(3, "pass");
System.out.println(sb); //123pass456
```

Accepts all primitive types and char[]

- `delete(int startIndex, int endIndex)` removes a substring within two indexes.

```
StringBuilder sb = new StringBuilder("123pass456");
sb.delete(3, 7);
System.out.println(sb); //123456
```

Exclusive the last index

StringBuilder Class (2)

- `.replace(int startInd, int endInd, String str)`

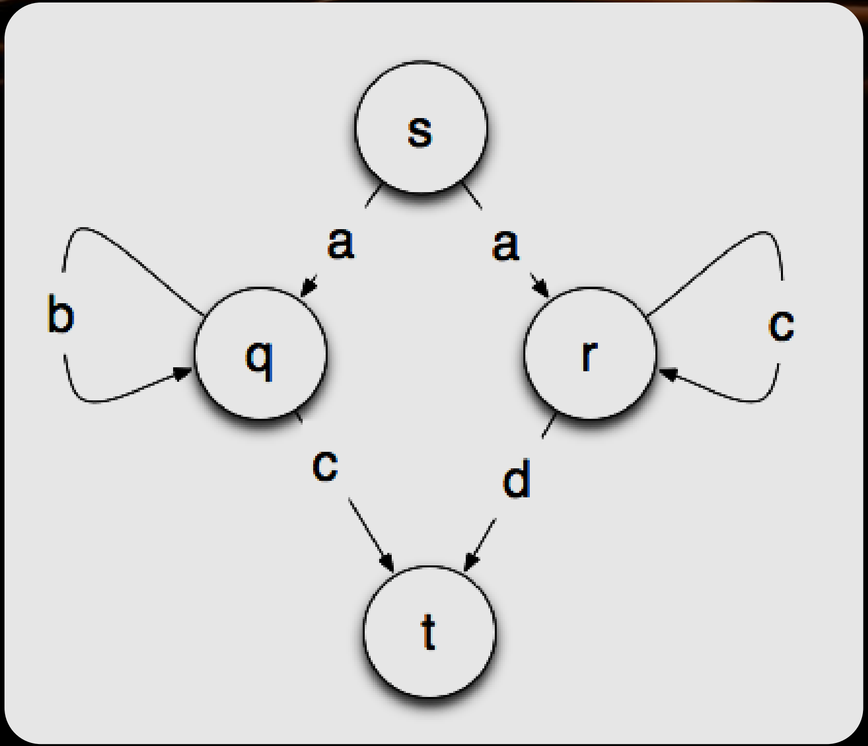
```
StringBuilder sb = new StringBuilder("123pass456");
sb.replace(3, 7, "woo");
System.out.println(sb);  //123woo456
```

- `.reverse()` - replaces a String by a reversed copy of it.

```
StringBuilder sb = new StringBuilder("123456");
sb.reverse();
System.out.println(sb);  //654321
```

Practice: String Manipulations

Live Exercises in Class (Lab)

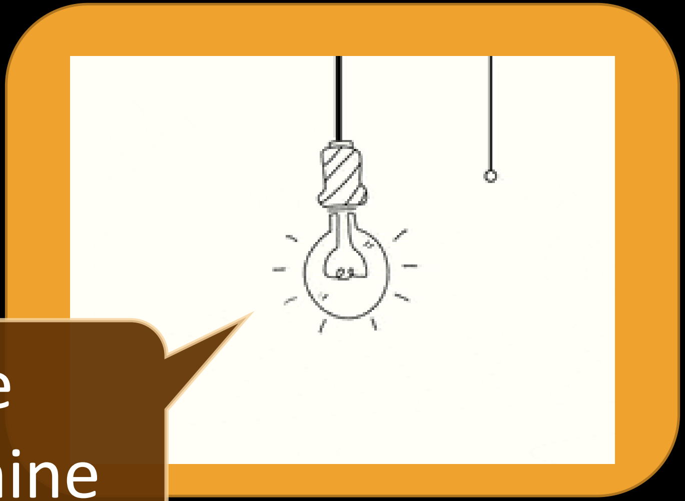

$$\begin{tikzcd}
 & s \arrow[dl, "a"'] \arrow[dr, "a"] & \\
q \arrow[loop left, "b"] \arrow[dr, "c"'] & & r \arrow[loop right, "c"] \arrow[dl, "d"] \\
 & t &
\end{tikzcd}$$

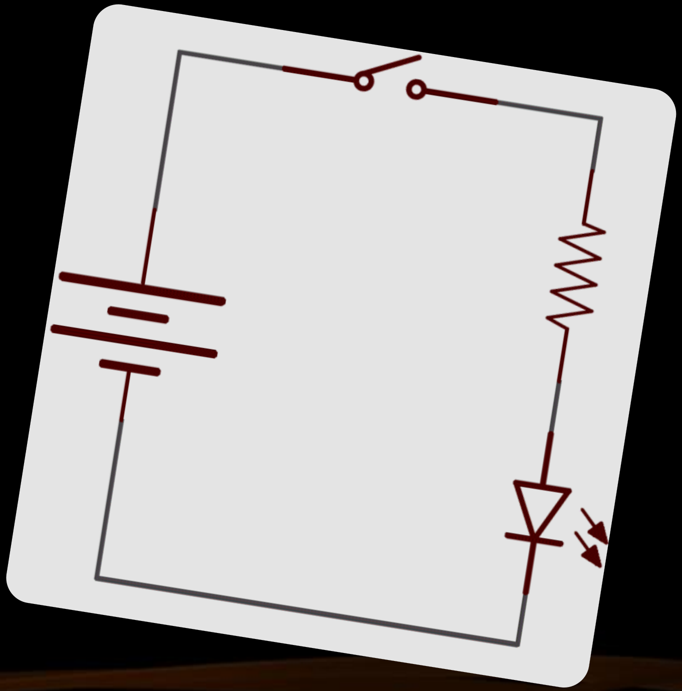
State Machines

Real life examples of a state machine

State Machine

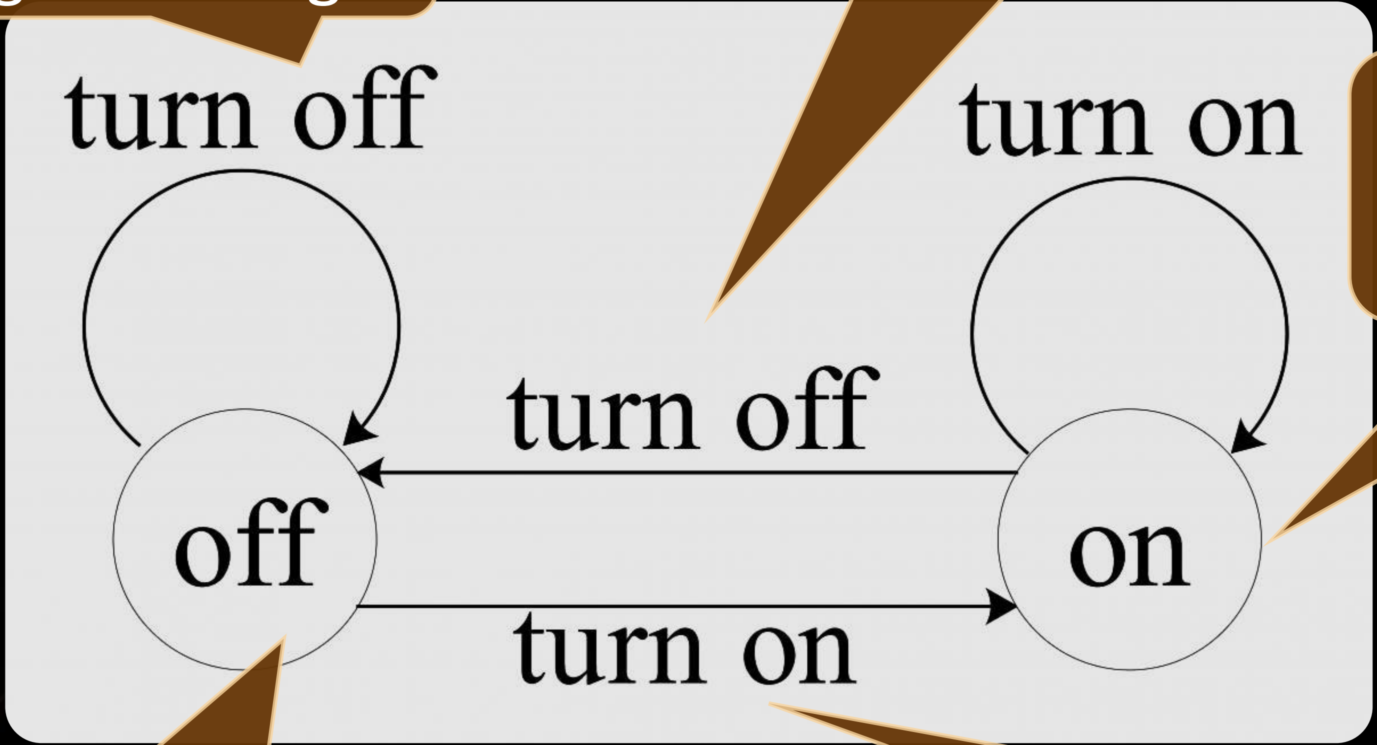
- Conceptual **model** used to describe how things work
  - Every time it reads an input it will **switch** to a different **state**
  - Only **one** state can be **active** at the same time
  - Each **state** specifies which state to **switch next**





A Lightbulb is real life example of a state machine

Example of a Finite State Machine (FSM)


$$\begin{tikzcd}
\text{off} \arrow[loop above, "\text{turn off}"] \arrow[r, bend right, "\text{turn on}"'] & \text{on} \arrow[loop above, "\text{turn on}"] \arrow[l, bend right, "\text{turn off}"']
\end{tikzcd}$$

Pressing **OFF** changes nothing

We can **return** to previous state

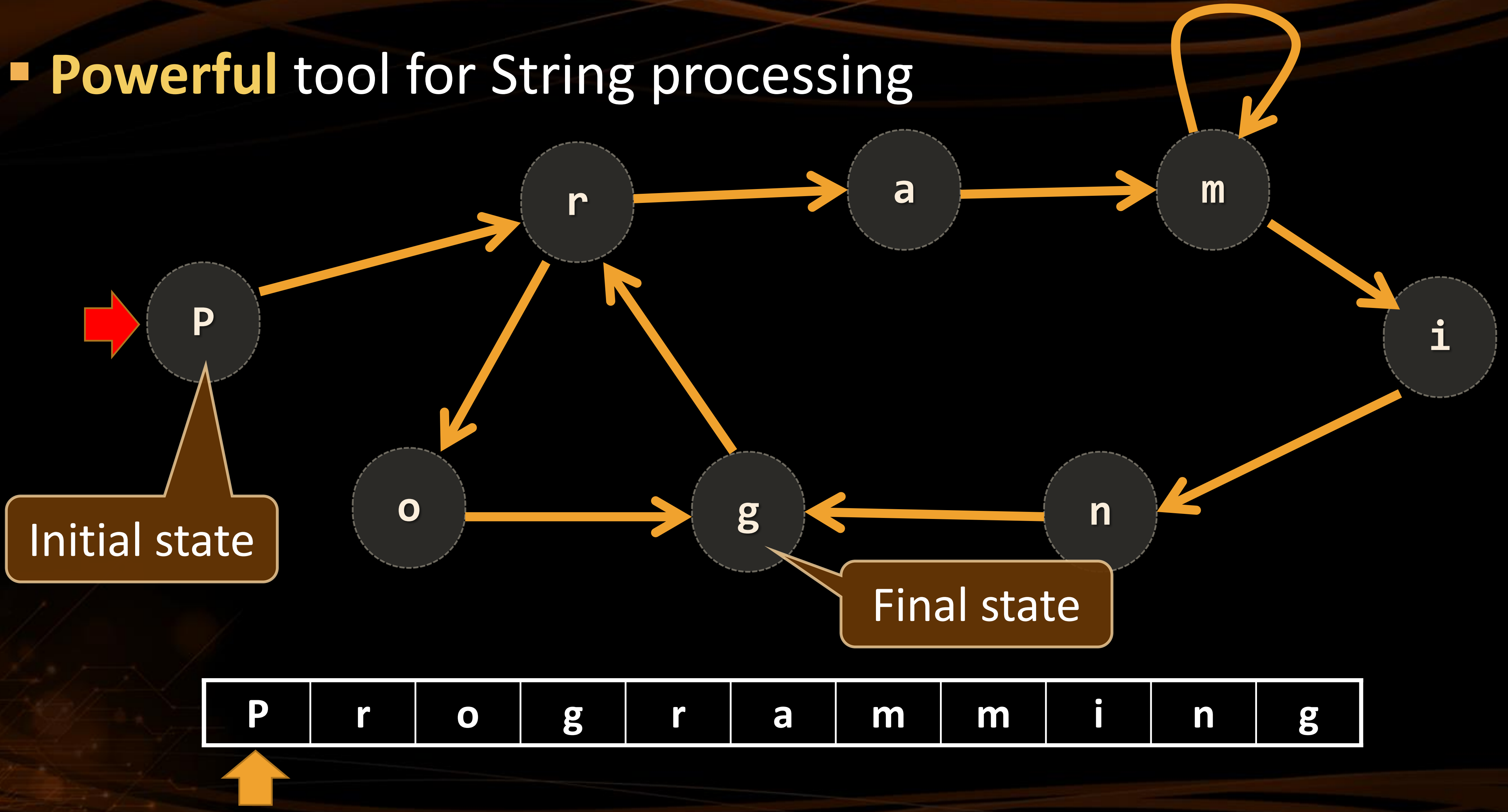
Finite = limited number of states available

Start state - **OFF**

Pressing **ON** changes the current state

State Machine for String Processing

- **Powerful** tool for String processing



State Machine for String Processing (2)

```
int state = 0;
int i = 0;
while (i <= input.length) {
   switch (state) {
      case 0:     //Initial state
         if (input[i] == 'p' || input[i] == 'P')  state = 1;
         else state = -1;
         break;
      case 1:    // P or p
         if (input[i] == 'r') state = 2;
         else state = -1;
         break;
      case 2:   // r
         if (input[i] == 'o') state = 3;
         else if (input[i] == 'a') state = 5;
         else state = -1;
         break;
```

The **initial** state is always reached

Multiple transitions from a state are possible

State Machine for String Processing (3)

```
case 3:   // o
   if (input[i] == 'g') state = 4;
   else state = 10;
   break;
case 4:   // g
   if (i == 11) {      //End state
      System.out.println("Word is valid"); return;
   else if (input[i] == 'r') state = 2;
   else state = -1;
   break;
case 5:   // a
   if (input[i] == 'm') state = 6;
   else state = -1;
   break;
```

There is a case for each state

If the **final** state is reached – the word is valid

State Machine for String Processing (4)

```
      case 6:   // m
         if (input[i] == 'm') break;
         else if (input[i] == 'i') state = 7;
         else state = -1;
         break;
      case 7:   // i
         if (input[i] == 'n') state = 8;
         else state = -1;
         break;
      case 8:   // n
         if(input[i] == 'g') state = 4;
         else state = -1;
         break;
      default:
         System.out.println("The word is not valid");
         break;
   } ++i; }
```

The **default** case handles **invalid** input

Problem: Series of Letters

- Read a String from the console
- **Replace** series of **consecutive identical letters** with a **single one**
- Solve the problem building your own **state machine**

$$\begin{tikzcd} \texttt{bookkeeper} \arrow[d] & \texttt{tattoo} \arrow[d] \\ \texttt{bokeper} & \texttt{tato} \end{tikzcd}$$

Check your solution here: https://judge.softuni.bg/Contests/777

Series of Letters: Solution with FSM

```
int state = 0; char prev = input[0];
   for (int i = 0; i < input.length; i++){
     switch (state){
       case 0:      //Initial state
         state = 1;
         prev = input[i]; break;
       case 1:     // Found a new letter
         output.append(prev);
         if (input[i] == prev)
           state = 2;
         prev = input[i]; break;
```

Check your solution here: https://judge.softuni.bg/Contests/777

Series of Letters: Solution with FSM (2)

```
     case 2:   // Found the same letter
       if (input[i] != prev)
         state = 1;
       prev = input[i]; break;
}

if(input[i-2] != prev)
   output.append(prev);

System.out.println(output)
```

Check your solution here: https://judge.softuni.bg/Contests/777

```
(?<=\.) {2,}(?=[A-Z])
```

Regular Expressions

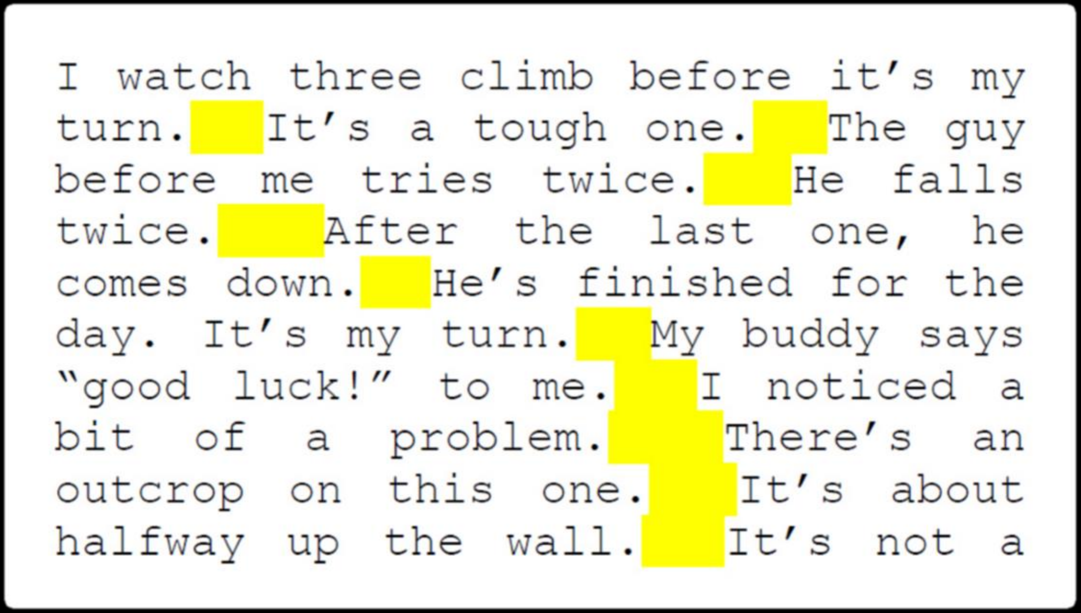
Using RegEx in Java

Regular Expressions

- Sequence of characters that forms a search **pattern**

```
(?<=\.) {2,}(?=[A-Z])
```

- Used for **finding and matching** certain parts of strings
- Most common application of a **finite state machine**



I watch three climb before it's my turn. It's a tough one. The guy before me tries twice. He falls twice. After the last one, he comes down. He's finished for the day. It's my turn. My buddy says "good luck!" to me. I noticed a bit of a problem. There's an outcrop on this one. It's about halfway up the wall. It's not a

Regex in Java

- Regex in Java library
  - `java.util.regex.Pattern`
  - `java.util.regex.Matcher`

```java
Pattern pattern = Pattern.compile("a*b");
Matcher matcher = pattern.matcher("aaaab");

boolean match = matcher.find();
String matchText = matcher.group();
```

Searches for the next match

Gets the matched text

Validating String By Pattern

- `Pattern.matches(String pattern, String text)` – determines whether the text matches the pattern

```
String text = "Today is 2015-05-11";
String pat = "\\d{4}-\\d{2}-\\d{2}";

boolean containsValidDate =
     Pattern.matches(pat, text);

System.out.print(containsValidDate); // true
```

Shorthand for [0-9]

Checking for a Single Match

- find() - Gets the first pattern match

Matches the element **one or more times**

```
String text = "Andy: 123";
String pattern = "([A-Z][a-z]+): (\\d+)";

Pattern regex = Pattern.compile(pattern);
Matcher matcher = regex.matcher(text);

matcher.find();
```

Group 0 = **Andy: 123**
Group 1 = **Andy**
Group 2 = **123**

Splitting With Regex

- `split(String pattern)` – splits the text by the pattern
  - Returns `String[]`

```java
String text = "1   2 3      4";
String pattern = "\\s+";

String[] tokens = text.split(pattern);
```

Matches whitespaces

tokens = { "1", "2", "3", "4" }

Series of Letters: Solution with RegEx

```
public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        String pattern = "([a-zA-Z ])\\1*

        Pattern regex = Pattern.compile(pattern);
        Matcher matcher = regex.matcher(input);

        while (matcher.find()) {
            System.out.print(matcher.group(1));
        }
    }
```

Matches the value of **group 1**

Check your solution here: https://judge.softuni.bg/Contests/777

Helpful Resources

- https://regex101.com and http://regexr.com – websites to test Regex using different programming languages
- http://docs.oracle.com/javase/7/docs/api/java/util/regex/Matcher – a quick reference for Regex from Oracle
- http://regexone.com – interactive tutorials for Regex
- http://www.regular-expressions.info/tutorial.html – a comprehensive tutorial on regular expressions

Problem: Vowel Count

SoftUni Foundation

- Find the count of all vowels in a given text
  - vowels are upper and lower a, e, i, o, u and y

```
Abraham Lincoln
```

Vowels: 5

```
In 1519 Leonardo da Vinci died at
the age of 67.
```

Vowels: 15

Check your solution here: https://judge.softuni.bg/Contests/777

Solution: Vowel Count

```java
String text = reader.readLine();
Pattern pattern =
     Pattern.compile("[AEIOUYaeiouy]");
Matcher matcher = pattern.matcher(text);

int count = 0;
while (matcher.find())
  count++;

System.out.println("Vowels: " + count);
```

Check your solution here: https://judge.softuni.bg/Contests/777

Problem: Extract Tags

- Extract all tags from a given HTML
- Read until an END command

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
</html>
END
```

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>
</title>
</head>
</html>
```

Check your solution here: https://judge.softuni.bg/Contests/777

Solution: Extract Tags

```
Pattern pattern = Pattern.compile("<.*?>");
String text = reader.readLine();

while (!text.equals("END")) {
  Matcher matcher = pattern.matcher(text);
  while (matcher.find())
    System.out.println(matcher.group());

  text = reader.readLine();
}
```

Matches the element **zero or one times**

Check your solution here: https://judge.softuni.bg/Contests/777

Problem: Valid Usernames

- Scan through the lines for **valid usernames**:
  - Has length **between 3 and 16** characters
  - **Contains** letters, numbers, hyphens and underscores
  - Has **no redundant symbols** before, after or in between

```
sh
too_long_username
!lleg@l ch@rs
jeff_butt
END
```

```
invalid
invalid
invalid
valid
```

Check your solution here: https://judge.softuni.bg/Contests/777

Solution: Valid Username

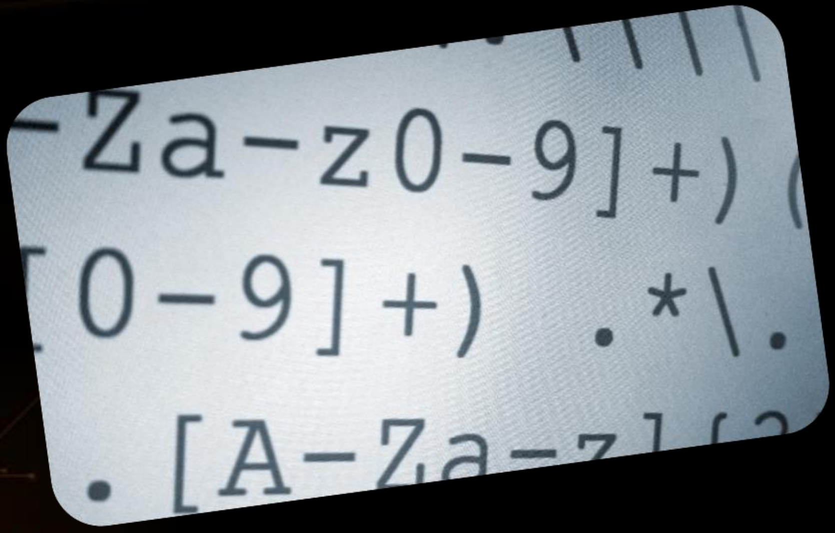
Match must start at the beginning of the string

```
Pattern pattern =
      Pattern.compile("^[a-zA-Z0-9_-]{3,16}$");
String text = reader.readLine();
while (!text.equals("END")) {
  Matcher matcher = pattern.matcher(text);
  if (matcher.find())
    System.out.println("valid");
  else
    System.out.println("invalid");

  text = reader.readLine();
}
```

Match must occur at the end of the string

Check your solution here: https://judge.softuni.bg/Contests/777



Practice: State Machines and Regex

Exercises in class

Summary

- Strings are immutable sequences of chars (instances of `java.lang.String`)
  - Can't be iterated
  - Support operations such as `substring()`, `indexOf()`, `trim()`, etc.
  - Changes to the String create a new object, instead of modifying the old one
- `StringBuilder` offers good performance
  - Recommended when concatenating Strings in a loop

Summary (2)

- State machines describe **how things work**
  - Often used for **String processing**
- **Regular expressions** describe **patterns** for searching through text
- They define special characters, operators and constructs
- Powerful tool for **extracting** or **validating data**
- Java provides a built-in **RegEx** classes

String Processing

SoftUni Foundation

xssoftware

SmartIT

NETPEAK
SEO and PPC for Business

SUPERHOSTING.BG

INDEAVR
Serving the high achievers

telenor

SOFTWARE GROUP

INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

Questions?

https://softuni.bg/courses/

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

Software University

SoftUni Foundation

FORUM