Java Course
Lecture 6 - Strings

PRAGMATIC

IT Learning & Outsourcing Center

www.pragmatic.bg

Summary

- String
  - String Processing
  - StringBuilder / StringBuffer
  - Converting between numbers and strings

What Is a String?

- *Strings* are sequences of characters
  - Represented by the String class
- A *String* object holds a sequence of characters
- *String* objects are read-only (immutable)
  - Their values cannot be changed after creation (this is by design behaviour)
- The String class represents all strings in Java

Creating a new string

- Using the string literal - double-quoted constant

  String lastName = “John";

- Concatenate strings:

  String fullName = firstName + " " + lastName;

- Use a constructor:

  String fullName = new String(“John Smith");

Concatenating Strings

- Use the + operator to concatenate strings

```
System.out.println("Name = " + name);
```

- You can concatenate primitives and strings

```
int age = getAge();
System.out.println("Age = " + age);
```

- String.concat() is another way to concatenate strings , it behaves the same ways as the + operator

```
System.out.println("Name = " .concat(name));
```

String Operations

- How to find the length of a string:
  - Use the length() method

```
String str = “John";
int len = str.length(); // len = 4
```

- How to find the character at a specific position:
  - Use the charAt(index) method
- Positions are counted from 0 to length()-1

```
String str = “John";
char c = str.charAt(1); // c = ‘o'
```

String Operations

- How to extract a substring of a string:
  - Use the following method :

```
String substring(int beginIndex, int endIndex);
```

  - The symbol at the position endIndex is not part of the result!
- Example :

```
String s = "How are strings processed in Java?";
String substr = s.substring(8,15); // strings
```

String Operations

- How to find the index of a substring

```
int indexOf(String str);
int lastIndexOf(String str);
```

- Examples:

```
String str = "Java is the best language ever .. yes Java !";
System.out.println(str.indexOf("Java"));  //  0
System.out.println(str.indexOf("best")); // 12
System.out.println(str.indexOf("Eclipse")); // -1
System.out.println(str.lastIndexOf("Java"));  // 38
```

Comparing Strings

- Use equals() to perform case-sensitive compare

```
String passwd = connection.getPassword();
if (passwd.equals("fgHPUw"))… // Case is important
```

- Use equalsIgnoreCase()if you want to ignore the case:

```
String category = getCategory();
if (category.equalsIgnoreCase("Movie")) ...
// We just want the word to match
```

- The == operator compares the references of the String objects
- The .equals(…) method compares the contents of the strings
- This example shows the difference:

```
String text1 = new String("some string");
String text2 = new String("some string");
boolean incorrectCompare = (text1 == text2); // false
boolean correctCompare = (text1.equals(text2)); // true
```

Empty and null Strings

- The String objects can have a value of **null**
  - Remember strings are Objects not primitives

```
String text = null;
```

- The empty string in not a **null** string

```
String empty = "";
```

- Calling methods of a **null** string causes *NullPointerExceptipon*

```
String s = null;
String empty = "";
boolean equal = s.equals(empty);
// NullPointerExceptipon will be thrown
```

Splitting Strings

- Use the method:

```
String[] split(String regularExpression)
```

Example:

```
String listOfBeers = "Amstel, Zagorka, Tuborg, Becks.";
String[] beers = listOfBeers.split("[,\\.]+");
System.out.println("Available beers are:");
for (String beer : beers) {
    System.out.println(beer);
}
```

- *You need to have a basic understanding of regular expressions*

What Are Regular Expressions?

- *Regular expressions* are a way to describe a set of strings based on common characteristics shared by each string in the set. They can be used to search, edit, or manipulate text and data.
- Example :

```
[0-1]+
```

- Matches all the strings consisting of the digits 0 and 1 (with a length at least 1)

```
088[0-9]{7}
```

- All phone numbers that have the format 088XXXXXXX (X is a digit)

Examples

- Regular expression like this :

  eclipse

  This means this exact 7 letters. Lowercase.

  [abc] Any of the letters a, b or c.

  [^abc] any except a b or c

  [a-zA-Z0-9] any letters between a to z + uppercase A-Z + digits 0-9

  beer|vodka exactly beer or vodka

Predefined characters

`.` – any character except a line terminator

`\d` – a digit same as `[0-9]`

`\D` – a non-digit same as `[^0-9]`

`\s` – a whitespace character

`\S` – a non-whitespace character

`\w` – a word character

`\W` – a non-word character

`* means 0` or more occurrences

`+ means one or more`

`? means 0 or 1`

- Positive Integers:

```
[1-9][0-9]*
```

- English word:

```
[a-zA-Z]+
```

- Phone number in Sofia

```
(02)?[0-9]{7}
```

Ok.. But how to use them in Java

- Easiest way to use :
- String.matches(regex) – check if whole string matches this regex
- String.split(regex) – splits a string based on regex
- String.replaceFirst() / replaceAll() –replaces one or all matches of some regular expression with another string

```
String num = "-127"; // Valid number
String patternIntNumber = "0|[+-]?[1-9][0-9]*";
boolean valid = num.matches(patternIntNumber);
```

What Are Regular Expressions?

- There are 3 core classes to work with regular expressions in java.util.regex package
  - Pattern - object is a compiled representation of a regular expression.
  - Matcher -object is the engine that interprets the pattern and performs match operations against an string.
  - PatternSyntaxException

- Pattern – holds compiled regular expression

```
Pattern pattern = Pattern.compile(regex);
```

- Matcher – performs matching/searches in the text

```
Matcher matcher = pattern.matcher(text);
while (matcher.find()) {
  // Process the matched substring
}
```

Full example with pattern and Matcher

```
String regex = "\\w+";
String text = "Hello, World!";
Pattern pattern = Pattern.compile(regex);
Matcher matcher = pattern.matcher(text);
while (matcher.find()) {
   int start = matcher.start();
   int end = matcher.end();
   String match = text.substring(start, end);
   System.out.println(match);
}
// Result:
// Hello
// World
```

Object.toString()

- Use the Object.toString() to convert the current instance into string
- Your class can override toString()
- System.out.println() automatically calls an object's toString() method when a reference is passed to it

String.valueOf()

- Use String.valueOf(): to convert a primitive to a string

```
String seven = String.valueOf(7);
```

Constructing Strings

- Strings are immutable
  - concat(), replace(), trim(), ... return new string, do not modify the old one
- Do not use "+" for strings in a loop!
  - It runs very inefficiently!

```
public String countChars(char ch, int count) {
    String result = "";
    for (int i = 0; i < count; i++)
        result += ch;

    return result;
}
```

StringBuilder / StringBuffer

- Use the StringBuilder class for modifiable strings of characters:

```
public String reverseIt(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

StringBuilder methods

- StringBuilder(int capacity) constructor allocates in advance buffer memory of a given size
  - By default 16 characters are allocated
- capacity() returns the currently allocated space (in characters)
- length() returns the length of the string
- charAt(int index) returns the char value at given position
- setCharAt(int index, char ch) changes a single character

StringBuilder methods

- append(…)
  - appends string or other type after the last character in the buffer
- delete(int start, int end)
  - removes the characters in given range
- insert(int offset, String str)
  - inserts given string at given position
- replace(int start, int end, String str)
  - replaces a substring by a given string
- toString()
  - converts the StringBuilder to String object

Converting

- From string to number
```
float a = Float.parseFloat("2.3");
```
- From number to string
```
int i = 5;
String s1 = "" + i;
//or
String s2 = String.valueOf(i);
//or
String s3 = Integer.toString(i);
```

Q and A ?

Problems

- How can we create a string object ?
- What's the difference between a string a char array ? Why not use char arrays instead of Strings ?
- Describe the most important classes and methods that allow using regular expressions in Java
- What's the difference between StringBuilder and String ?
- Write a program that finds how many times a word is found in a text. For example given text "You are awesome, but do you know how much awesome?" contains the word "awesome" two times

- Write a program that takes as input a list of words and a text and replaces in this text each of the words with asterisks. Example:

  words = "*Java, programming, course*"

  text = "*Welcome to Java programming language. This course will teach you programming for the Java platform.*"

  result = "*Welcome to* **** *********** *language. This* ****** *will teach you* *********** *for the* **** *platform.*"

  Hint: Extract the list of words and use `String.replaceAll(...)` for each word.