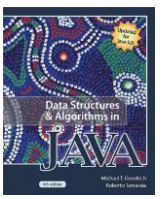
CSC9UT4 (Managing Information): String Manipulation in Java

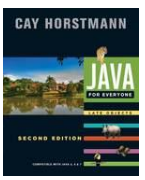
Jingpeng Li

http://www.cs.stir.ac.uk/~jli/

Content

- String manipulation in Java
- The use of files in Java
- References:
  - Java For Everyone, 2nd Edition (2013), by Cay Horstmann
    - Chapter 02: section 2.3 and 2.5 (on Strings)
    - Chapter 07: sections 7.1, 7.2, 7.3 (on Files)
  - M. T. Goodrich and R. Tamassia, Data Structures and Algorithms in Java, 5th edition,





- Character strings (such as those displayed in the board) are important data types in any Java program. We will learn how to work with text, and how to perform useful tasks with them.
- You will also learn how to write programs that manipulate text files, a very useful skill for processing real world data.

2

Strings

- Many programs process text, not numbers
- Text consists of characters: letters, numbers, punctuations, spaces, etc.
- What is a string?
  - An ordered collection of characters of arbitrary length
  - Consider Stirling, You rock!, “67.435”, 3.8x104, or 5.2e6

3

Common Uses of Strings

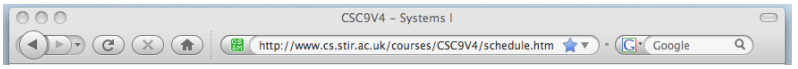
- Input
  - from a user, or from a (data) file
  - the program must understand what the string represents
  - making sense of a string (determining its syntax) is called *parsing* *e.g. a Url*
- Output
  - maybe on the screen, or to a file
- Strings are often converted to/from other formats, e.g.
  - string/number conversions are very common, including integers, floating point numbers
  - can also have general string/object conversions

As programmers, we need to be able to process strings!

4

Examples of String Processing

- A compiler takes the text of a program as input, and its first task is to parse the input
- A word processor looks to see which are the individual words of a line of text so that it can spell check them.
- A browser must parse a URL into pieces:
  - which protocol to use
  - which web server to contact
  - which file to ask for from that web server



5

Syntax of Input Statement

- The `Scanner` class allows you to read keyboard input from the user
  - It is part of the Java API `util` package

Java classes are grouped into packages. Use the `import` statement to use classes from packages.

Include this line so you can use the Scanner class.

```
import java.util.Scanner;
.
.
Scanner in = new Scanner(System.in);
.
.
System.out.print("Please enter the number of bottles: ");
int bottles = in.nextInt();
```

Create a Scanner object to read keyboard input.

Don't use println here.

Display a prompt in the console window.

Define a variable to hold the input value.

The program waits for user input, then places the input into the variable.

Page 6

Strings

- The `String` Type:
  - Type Variable Literal
  - `String name = "Harry"`
- Once you have a `String` variable, you can use methods such as:

```
int n = name.length();  // n will be assigned 5
```

- A `String`'s length is the number of characters inside:
  - An empty `String` (length 0) is shown as ""
  - The maximum length is quite large (an `int`)

String Processing Facilities in Java

- A common method to manipulate strings in Java is to use the **`String`** class (you can also use **`StringBuffer`**)
- The **`String`** class provides a lot of useful methods, including those for
  - creating and manipulating strings
  - inspecting the characters in a string
  - splitting up a string into tokens
- Some of the most useful of these
  - **`substring, trim, split`**
  - **`toUpperCase, toLowerCase`**
  - **`equals, endsWith, startsWith`**
  - **`charAt, indexOf`**, **`lastindexOf`**
- We will also look at the **`StringTokenizer`** class

TIP: Refer to Java docs! http://www.cs.stir.ac.uk/doc/java/jdk1.6/

String Concatenation

- Java: '+' operator is used to concatenate strings. Put them together to produce a longer string. Example:
  - `String fName = "Harry", String lName = "Morgan"`
  - `String name = fName + lName`
  - Results in the string: `"HarryNorman"`
- If you'd like the first and last name separated by a space:
  - `String name = fName + " " + lName`
  - Results in the string: `"Harry Norman"`
- When the expression to the left or right of a '+' operator is a string, the other one is automatically forced to be a string, and both strings are concatenated. Example:
  - `String jobTitle = "Agent", int empID = 7`
  - `String bond = JobTile + empID`
  - Results in the string: `"Agent7"`

9

Converting Strings to Numbers

- Strings can contain *digits*, not *numbers*
  - They must be converted to numeric types
  - 'Wrapper' classes provide a `parseInt` method

'3'	'0'	'3'	'8'	'2'	'4'	'6'	'4'	'6'

```
String pop = "303824646";
int populationValue = Integer.parseInt(pop);
```

- **Caution:**
  - The argument must be a string containing only digits without any additional characters. Not even spaces are allowed! So use the `trim` method before parsing!

```
int populationValue = Integer.parseInt(pop.trim());
```

Page 10

String Input

- Reading a String from the console:

```
System.out.print("Please enter your name: ");
String name = in.next();
```

  - The next method reads one word at a time
  - It looks for 'white space' delimiters
- Reading an entire line:

```
System.out.print("Please enter your address: ");
String address = in.nextLine();
```

  - The nextLine method reads until the user hits 'Enter'
- Converting a String variable to a number

```
System.out.print("Please enter your age: ");
String input = in.nextLine();
int age = Integer.parseInt(input);  // only digits!
```


String Escape Sequences

- How would you print a double quote?
  - Preface the " with a \ inside the double quoted String

```
System.out.print("He said \"Hello\"");
```

- OK, then how do you print a backslash?
  - Preface the \ with another \

```
System.out.print("“C:\\Temp\\Secret.txt“);
```

- Special characters inside Strings
  - Output a newline with a '\n'

```
System.out.print("*\n**\n***\n");
```

```
*
**
***
```

Strings and Characters

- Strings are sequences of characters
  - Characters have their own type: char
  - Characters have numeric values
    - See the ASCII code chart in Appendix B
    - For example, the letter 'H' has a value of 72 if it were a number

- Use single quotes around a char

```
char initial = 'B';
```

- Use double quotes around a String

```
String initials = "BRL";
```


Copying a char from a String

- Each char inside a String has an index number:

0	1	2	3	4	5	6	7	8	9
c	h	a	r	s		h	e	r	e

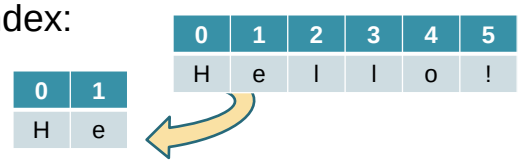
- The first char is index zero (0)
- The charAt method returns a char at a given index inside a String:

```
String greeting = "Harry";
char start = greeting.charAt(0);
char last = greeting.charAt(4);
```

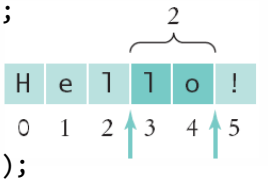
0	1	2	3	4
H	a	r	r	y

Copying portion of a `String`

- A substring is a portion of a `String`
- The `substring` method returns a portion of a `String` at a given index for a number of `chars`, starting at an index:




```
String greeting = "Hello!";
String sub = greeting.substring(0, 2);
//starts at 0, ends before 2
```




```
String sub2 = greeting.substring(3, 5);
```


Example: initials.java

```
1  import java.util.Scanner;
2
3  /**
4     This program prints a pair of initials.
5  */
6  public class Initials
7  {
8     public static void main(String[] args)
9     {
10       Scanner in = new Scanner(System.in);
11
12       // Get the names of the couple
13
14       System.out.print("Enter your first name: ");
15       String first = in.next();
16       System.out.print("Enter your significant other's first name: ");
17       String second = in.next();
18
19       // Compute and display the inscription
20
21       String initials = first.substring(0, 1)
22          + "&" + second.substring(0, 1);
23       System.out.println(initials);
24    }
25 }
```

The `StringTokenizer` Class

- The StringTokenizer class allows a string to be split into pieces know as '*tokens*'
- A *delimiter* character is specified, and this is used to break down the original string into tokens. We start a new token every time a delimiter character is detected.
- For example, with the string
  `"http://www.cs.stir.ac.uk/courses/CSC9V4/"`
  and the delimiter '/', the individual tokens in that string are `"http:"`, `"www.cs.stir.ac.uk"`, `"courses"`, and `"CSC9V4"`.
- More usually, with the default space character delimiter, a line of text is broken up into individual words. This is how a word processor works out where words begin and end.

17

`StringTokenizer` Example 1

```
StringTokenizer st; // Declare a reference
st = new StringTokenizer("this is a test");

while (st.hasMoreTokens())
{
   System.out.println(st.nextToken());
}
```

The above program produces:

```
this
is
a
test
```

18

StringTokenizer Example 2

```
String
  input="http://www.cs.stir.ac.uk/index.htm";
String delims= "/";

StringTokenizer st;   // Declare a reference
st = new StringTokenizer(input,delims);

while (st.hasMoreTokens())
    System.out.println(st.nextToken());
```

The above program produces:

```
http:
www.cs.stir.ac.uk
index.htm
```

19

The String.split method

- In Java 1.5, a new method of tokenizing strings was introduced. An additional method called **split** was added to the **String** class.
- The API guide for **split** is:

  ```
  public String[] split(String regex)
  ```

  where regex is a 'regular expression' or pattern used to determine how to break up the String.
- For example we could just use the forward slash delimiter as before "/", alternatively we can use "\\s+" which means one or more white spaces, or "two\\s+" which looks for the word 'two' followed by one or more white spaces.
- Regular expressions are very powerful selectors
- **split** returns an array of tokens; each token is just a String

http://ocpsoft.org/opensource/guide-to-regular-expressions-in-java-part-1/

Example: `String.split`

```
String
  input="http://www.cs.stir.ac.uk/index.htm";
String delims="/";

String [] tokens = input.split(delims);

for (int t=0; t<tokens.length; t++)
   System.out.println(tokens[t]);
```

The above code produces:

```
http:
www.cs.stir.ac.uk
index.htm
```

21

String Operations (1)

Table 9 String Operations

Statement	Result	Comment
`string str = "Ja";` `str = str + "va";`	`str` is set to `"Java"`	When applied to strings, `+` denotes concatenation.
`System.out.println("Please"` `+ " enter your name: ");`	Prints `Please enter your name:`	Use concatenation to break up strings that don't fit into one line.
`team = 49 + "ers"`	`team` is set to `"49ers"`	Because `"ers"` is a string, 49 is converted to a string.

`String first = in.next();`
`String last = in.next();`
`(User input: Harry Morgan)`
	`first` contains `"Harry"`
	`last` contains `"Morgan"`
		The `next` method places the next word into the string variable.

`String greeting = "H & S";` `int n = greeting.length();`	`n` is set to 5	Each space counts as one character.
`String str = "Sally";` `char ch = str.charAt(1);`	`ch` is set to `'a'`	This is a `char` value, not a `String`. Note that the initial position is 0.

Page 22

String Operations (2)

Statement	Result	Comment
`String str = "Sally";` `String str2 = str.substring(1, 4);`	`str2` is set to `"all"`	Extracts the substring starting at position 1 and ending before position 4.
`String str = "Sally";` `String str2 = str.substring(1);`	`str2` is set to `"ally"`	If you omit the end position, all characters from the position until the end of the string are included.
`String str = "Sally";` `String str2 = str.substring(1, 2);`	`str2` is set to `"a"`	Extracts a `String` of length 1; contrast with `str.charAt(1)`.
`String last = str.substring(` `  str.length() - 1);`	`last` is set to the string containing the last character in `str`	The last character has position `str.length() - 1`.

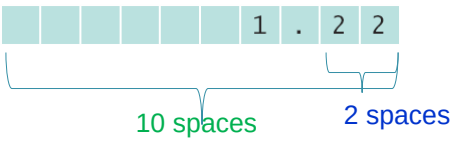
Formatted Output

- Outputting floating point values can look strange:

```
Price per liter:  1.21997
```

- To control the output appearance of numeric variables, use formatted output tools such as:

```
System.out.printf("%.2f", price);
  Price per liter: 1.22
System.out.printf("%10.2f", price);
  Price per liter:       1.22
```



- The `%10.2f` is called a format specifier

Formatting Output – cont.

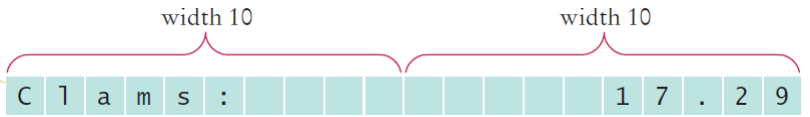
- Advanced `System.out.printf`
  - Can align strings and numbers
  - Can set the field width for each
  - Can left align (default is right)

```
Cookies:      3.20
Linguine:     2.95
Clams:       17.29
```

- Two format specifiers example:

```
System.out.printf("%-10s%10.2f", items[i] + ":", prices[i]);
```

  - `%-10s` : Left justified String, width 10
  - `%10.2f` : Right justified, 2 decimal places, width 10



`printf` Format Specifier

- A format specifier has the following structure:
  - The first character is a %
  - Next, there are optional "flags" that modify the format, such as **-** to indicate left alignment
  - Next is the field width, followed by an optional precision for floating-point numbers
- The format specifier ends with the format type, such as **f** for floating-point values or **s** for strings.

```
System.out.printf("%-10s%10.2f", items[i] + ":", prices[i]);
```

printf Format Flags

Table 2 Format Flags

Flag	Meaning	Example
-	Left alignment	1.23 followed by spaces
0	Show leading zeroes	001.23
+	Show a plus sign for positive numbers	+1.23
(	Enclose negative numbers in parentheses	(1.23)
,	Show decimal separators	12,300
^	Convert letters to uppercase	1.23E+1

Page 27

printf Format Types

- Formatting is handy to align columns of output

Table 8 Format Types

Code	Type	Example
d	Decimal integer	123
f	Fixed floating-point	12.30
e	Exponential floating-point	1.23e+1
g	General floating-point (exponential notation is used for very large or very small values)	12.3
s	String	Tax:

- You can also include text inside the quotes:

```
System.out.printf("Price per liter: %10.2f", price);
```

Page 28

Summary: Strings

- **Strings** are sequences of characters.
- The length method yields the number of characters in a String.
- Use the + operator to concatenate Strings; that is, to put them together to yield a longer String.
- Use the next (one word) or nextLine (entire line) methods of the Scanner class to read a String.
- Whenever one of the arguments of the + operator is a String, the other argument is converted to a String.
- If a String contains the digits of a number, you use the Integer.parseInt or Double.parseDouble method to obtain the number value.
- String index numbers are counted starting with 0.
- Use the substring method to extract a part of a String

Next Lectures

- Reading and Writing Text Files
- Text Input and Output
- Command Line Arguments