



Text Processing In Java: Characters and Strings

Wellesley College CS230
Lecture 03
Monday, February 5
Handout #10
(Revised Friday, February 9)

Reading:
Problem Set:

Downey: Chapter 7
Assignment #1 due Tuesday, February 13

3 - 1



Characters

Characters are the fundamental unit of strings & files

Character literals:

- letters: 'a', 'b', ..., 'z', 'A', 'B', ... 'Z'
- digits: '0', '1', ..., '9'
- punctuation: ',', '.', ':', '!', ' ' (space), ...
- escaped characters:
 - '\"' (single quote)
 - '\"' (double quote)
 - '\\ ' (backslash)
 - '\n' (newline)
 - '\r' (carriage return)
 - '\t' (tab)
 - '\b' (backspace)
 - '\f' (form feed)

3 - 2



ASCII Character Representations

ASCII = American Standard Code for Information Exchange
Defines integer representations for $2^8 = 256$ characters
(see <http://www.asciitable.com>):

- lowercase letters: 'a' \leftrightarrow 97, 'b' \leftrightarrow 98, ..., 'z' \leftrightarrow 122
- uppercase letters: 'A' \leftrightarrow 65, 'B' \leftrightarrow 66, ..., 'Z' \leftrightarrow 90
- digits: '0' \leftrightarrow 48, '1' \leftrightarrow 49, ..., '9' \leftrightarrow 57
- punctuation: ' ' \leftrightarrow 32, '!' \leftrightarrow 33, '(' \leftrightarrow 41, ')' \leftrightarrow 42, ...
- escaped chars: '\n' \leftrightarrow 10, '\r' \leftrightarrow 13, '\"' \leftrightarrow 34, '\"' \leftrightarrow 39, '\\' \leftrightarrow 92

Use casts to convert between characters and integers:

- (int) 'A' \rightarrow 65, (int) 'a' \rightarrow 97
- (char) 65 \rightarrow 'A', (char) 97 \rightarrow 'a'

Can compare characters directly via <, <=, ==, !=, >=, >:

'a' < 'b' \rightarrow true, 'b' == 'b' \rightarrow true, 'c' <= '0' \rightarrow false

More modern Unicode representations define integer representations for $2^{16} = 65536$ characters; ASCII are the first 256 of these.

3 - 3



Putting ASCII Representations to Work

```
public static boolean isLetter (char c) {
    return ('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z'); }

public static boolean isDigit (char c) {
    return ('0' <= c && c <= '9'); }

public static boolean isUpperCase (char c) { // isLowerCase is similar
    return 'A' <= c && c <= 'Z'; }

public static char toLowerCase (char c) { // toUpperCase is similar
    return isUpperCase(c) ? (char) (((int) c) + 32) : c; }

public static String toString (char c) {
    return "" + c; }

public static char shiftChar (char c, int shift) {
    return (char) MathOps.mod(((int) c) + shift, 256);
} // MathOps.mod(x,m) returns x modulo m
```

3 - 4



Class Methods of the Character Class

Many character methods are already provided as class methods of the Character class, so we needn't define them from scratch. Moreover, these work on full unicode characters, not just ASCII subset. E.g.:

```
public static boolean isLetter (char c);
public static boolean isDigit (char c);
public static boolean isLetterOrDigit (char c);
public static boolean isWhitespace (char c);
public static boolean isUpperCase (char c);
public static boolean isLowerCase (char c);
public static char toLowerCase (char c);
public static char toUpperCase (char c);
public static String toString (char c);
```

See [Java API of the Character class](#) for the complete list of methods.

3 - 5



Java Strings vs. Character Arrays

```
char [] chars = {'c', 's', '2', '3', '0'};
String str = "cs230";
```

Like arrays of characters, Java strings are sequences of characters indexed from 0:

```
str.charAt(0) → 'c', ..., str.charAt(4) → '0'
chars[0] → 'c', ..., chars[4] → '0'
```

However, Java Strings differ from character arrays in important ways:

- Strings are instances of a class (String), but arrays are not instances of any class; can invoke String instance methods on strings.
- Strings are immutable --- cannot be changed once created.
- Different notations for String literals (double quotes) and accessing elements at an index (charAt() method).

3 - 6



Some String Instance Methods

```
String s1 = "delivery";  
s1.length() → 8  
s1.charAt(3) → 'i'  
s1.charAt(8) → StringIndexOutOfBoundsException  
s1.indexOf('e') → 1  
s1.indexOf('q') → -1 // Indicates character is not present  
s1.indexOf('e', 2) → 5  
s1.indexOf('e', 9) → -1 // *No* exception here!  
s1.indexOf("live") → 2  
s1.indexOf("love") → -1  
s1.indexOf("live", 9) → -1 // *No* exception here!
```

3 - 7



More String Instance Methods

```
s1.substring(4) → "very"  
s1.substring(8) → ""  
s1.substring(9) → StringIndexOutOfBoundsException  
s1.substring(2, 6) → "live"  
s1.substring(4, 8) → "very"  
s1.substring(5, 5) → ""  
s1.substring(6, 9) → StringIndexOutOfBoundsException  
s1.substring(7, 6) → StringIndexOutOfBoundsException  
s1.toUpperCase() → "DELIVERY"  
(s1.toUpperCase()).toLowerCase() → "delivery"  
s1.toCharArray() → {'d', 'e', 'l', 'i', 'v', 'e', 'r', 'y'}
```

3 - 8



Even More String Instance Methods

```
String s2 = "I say \t\"Yes\" \nyou say \t\"No\" \nI say \t\"Why?\"";
System.out.println(s2);
I say    "Yes"
you say          "No"
I say    "Why?"
s2.length() → 40      s2.charAt(6) → '\t'
s2.charAt(7) → '\"'    s2.charAt(12) → '\n'
s2.replace('s', 'S') → "I Say \t\"YeS\" \nyou Say \t\"No\" \nI Say \t\"Why?\"";
s2.replaceAll("say", "said") → "I said \t\"Yes\" \nyou said \t\"No\" \nI said \t\"Why?\"";
s2.replaceAll("[\t\n]", " ") // [\t\n] is a regular expression denoting a tab *or* a newline.
→ "I say \"Yes\" you say \"No\" I say \"Why?\"";
"aabaacadaaaae".replaceAll("a+", "XYZ") → "XYZbXYZcXYZdXYZe"
// a+ is a regular expression denoting any nonempty sequence of a's
"p \t \n q \n rs \t t".replaceAll("\\s+", " ") → "p q rs t"
// \s+ is a regular expression denoting any nonempty sequence of whitespace
// characters (e.g., spaces, tabs, newlines, carriage returns). The extra backslash
// is needed to pass the backslash before the s to the regular expression checker.
```

3 - 9



Yet More String Instance Methods

```
s2.split("\n")
→ {"I say \t\"Yes\" ", "you say \t\"No\" ", "I say \t\"Why?\""}
s2.split("[\n\t]");
→ {"I say ", "\"Yes\"", "you say ", "\"No\"", "I say ", "\"Why?\""}
s2.split("[\n\t ]"); // [\n\t ] is a regular expression denoting a
// newline *or* a tab *or* a space.
→ {"I", "say", "", "\"Yes\"", "you", "say", "", "\"No\"", "I", "say", "",
 "\"Why?\""}
// The empty strings come from three space/tab combinations in the input.
s2.split("\\s+"); // see notes on \s+ on the previous slide
→ {"I", "say", "\"Yes\"", "you", "say", "\"No\"", "I", "say", "\"Why?\""}
// Using \s+ removes the empty strings
"\t \na\t \nb\t \n".trim() → "a\t \nb"
```

How do we convert strings like "{ 17 , -3 , 42 }" to the corresponding integer arrays (e.g. {17,-3,42})? (See `IntArray.fromString()` in `utils` directory.)

See [Java API of the String class](#) for the complete list of String methods.

3 - 10



String Comparisons

```
"cs230".equals("cs230") → true  
"cs230".equals("CS230") → false  
"cs230" == "cs230" → Could be true, could be false!  
"prefer".compareTo("radar") → a negative integer  
"prefer".compareTo("preference") → a negative integer  
"prefer".compareTo("prefer") → 0  
"prefer".compareTo("like") → a positive integer
```

3 - 11



Simple String Methods

```
// Assume these are define in a StringOps class  
  
// first("cs230") → 'c'  
public static char first (String s) {  
    }  
  
// butFirst("cs230") → "s230"  
public static String butFirst (String s) {  
    }  
  
// last("cs230") → '0'  
public static char last (String s) {  
    }  
  
// butLast("cs230") → "cs23"  
public static String butLast (String s) {  
    }
```

3 - 12



String Mapping 1 (Use Iteration)

```
public static String toUpperCase (String s) {
```

```
}
```

How would we modify the above to implement the following?

```
public static String replace (String s, char oldChar, char newChar);
```

```
public static String shiftString (String s, int shift);
```

```
// StringOps.shiftString("HAL", 1) → "IBM"
```

3 - 13



String Mapping 2 (Use Recursion)

```
public static String toUpperCase (String s) {
```

```
}
```

3 - 14



String Filtering (Iterative and Recursive)

```
// StringOps.removeChar("delivery", 'e') → "dlivry"  
// StringOps.removeChar("delivery", 'd') → "elivry"  
// StringOps.removeChar("delivery", 'q') → "delivery"
```

```
public static String removeChar (String s, char c) {
```

```
}
```

3 - 15



String Reversal (Iterative and Recursive)

```
// StringOps.reverse("delivery") → "yreviled"  
public static String reverse (String s) {
```

```
}
```

3 - 16