Lecture 4
String in Java

Department of Computer Science
Hofstra University

Lecture Goals

- Describe how Strings are represented in Java Platform
- Perform basic operations with Strings in Java
- Work with the String's built-in methods to manipulate Strings
- Write regular expressions to match patterns and split strings

Motivation Example

There is hereby imposed on the taxable income of every individual (other than a surviving spouse as defined in section 2(a) or the head of a household as defined in section 2(b)) who is not a married individual (as defined in section 7703) a tax determined in accordance with the following table:

Hard to read

26 U.S. Code § 1 – Tax imposed
https://www.law.cornell.edu/uscode/text/26/1

How do we quantify the difference?

If you are single, never lost your spouse, and not the head of a household, you pay taxes according to the following table:

Easy to read

Use **flesch score** to measure of text readability

https://en.wikipedia.org/wiki/Flesch%E2%80%93Kincaid_readability_tests

Measure the Text Readability by Flesch Score

number of words per sentence

number of syllables per word

$$\text{FleschScore} = 206.835 - 1.015\left(\frac{\#\,\text{words}}{\#\,\text{sentences}}\right) - 84.6\left(\frac{\#\,\text{syllables}}{\#\,\text{words}}\right)$$

High score: Few words/sentence and few syllables/word

Low score: Many words/sentence and many syllables/word

longer word makes text harder to read than longer sentence

Document is represented as a big long string. Requires the ability to manipulate Strings!

Score	School level	Notes
100.00-90.00	5th grade	Very easy to read. Easily understood by an ave
90.0–80.0	6th grade	Easy to read. Conversational English for consume
80.0–70.0	7th grade	Fairly easy to read.
70.0–60.0	8th & 9th grade	Plain English. Easily understood by 13- to 15-year-old students.
60.0–50.0	10th to 12th grade	Fairly difficult to read.
50.0–30.0	College	Difficult to read.
30.0–0.0	College graduate	Very difficult to read. Best understood by university graduates.

There is hereby imposed on the taxable income of every individual (other than a surviving spouse as defined in section 2(a) or ... defined in section 2(b)) ... al (as defined in section 7703) a tax determined in accordance with the following table:

FleshScore = 12.6

If you are s..., and not the hea... axes according to the following table:

FleshScore = 65.8

String Basics

```
String text1 = new String("Hello World!");
String text2 = text1;
String text3 = text1.concat("It's a great day.");
String text4 = text1 + "It's a great day.";
String text5 = "Hello World!";
String text6 = "Hello World!";
String text7 = new String("Hello World!");
text7.equals(text1);          // true
text7 == text1;               // false
```

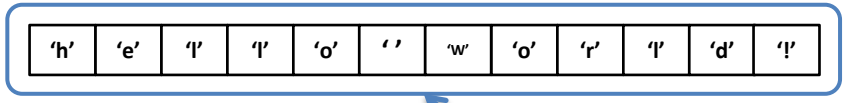
String is an object

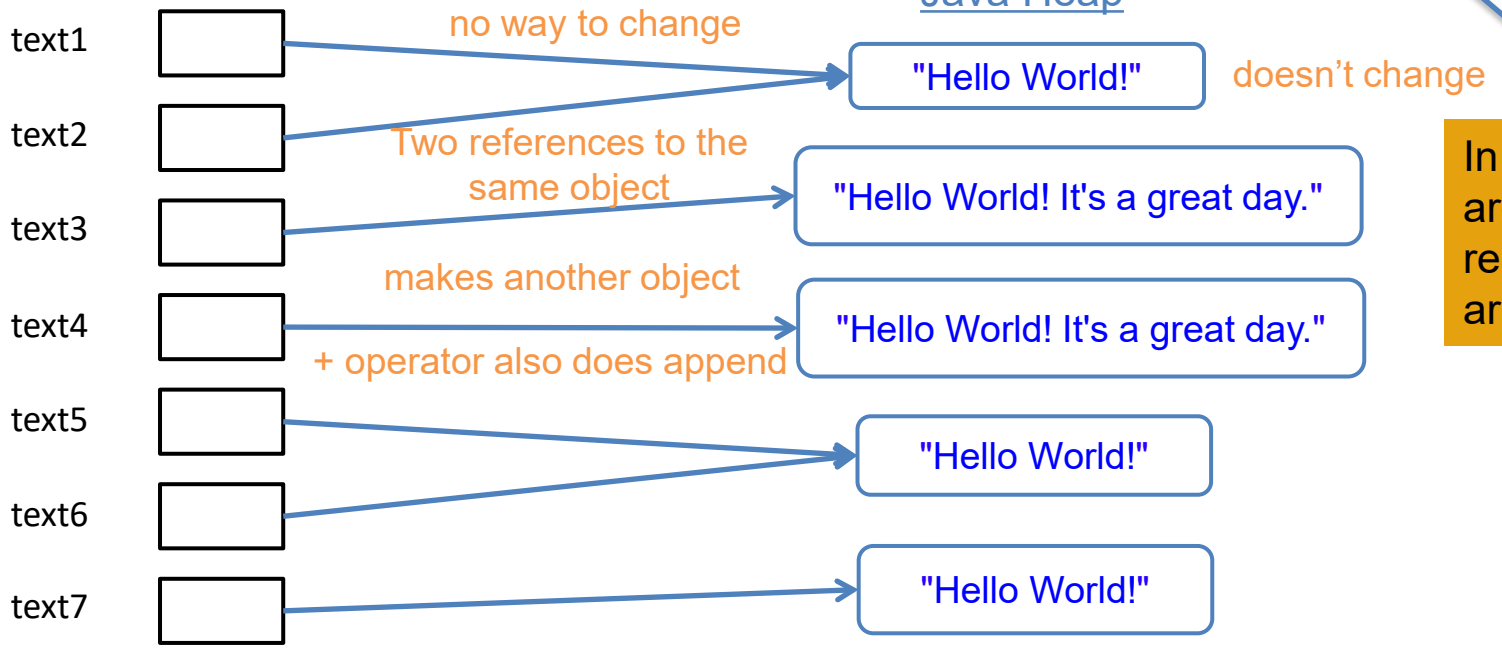
Strings are immutable

String append

Interned Strings: One object

Compare string





In heap, Strings are basically represented as arrays of chars

String Class's Built-in Methods

- Strings can do lots of things:
  - https://docs.oracle.com/javase/10/docs/api/java/lang/String.html
- Let's look at some methods in the context of our problems:
  - `length, charAt, toCharArray, indexOf, split`
- For example, we need to look at words, character by character, to calculate the number of syllables.

does the `letter` appear anywhere in the `word`?

```
public static boolean hasLetter(String word, char letter)
{
        for (int i = 0; i < word.length(); i++) {
                if (word.charAt(i) == letter) {
                        return true;
                }
        }
        return false;
}
```

`charAt(i)` cannot be used to change the String

If no letters match, return false

Loop over the indexes of character array in the string

`length()` returns the number of characters in the String

Get each letter and compare it to the char in question

`charAt(i)` returns the char at index i in the String

Count the number of syllables (Contd.)

```
public static boolean hasLetter(String word, char letter)
{
        for (char c: word.toCharArray()) {
                if (c == letter) {
                        return true;
                }
        }
        return false;
}
```

Same method, using a for-each loop

`toCharArray()` returns the chars in a String, as a char[]

Change this method so that it returns the index where it first finds letter (or -1 if it doesn't find it)?

```
int
public static boolean hasLetter(String word, char letter)
{
        for (int i = 0; i < word.length(); i++) {
                if (word.charAt(i) == letter) {
                        return true;
                                    i
                }
        }               -1
        return false;
}
```

built-in String method `indexOf(String str)` does exactly this, but with a String to match.

0

```
String text = "Can you hear me? Hello, hello?
int index = text.indexOf("he");  // index is 8
index = text.indexOf("He");      // index is 17
index = text.indexOf("Help");    // index is -1
```

For dealing with case, check out String methods:
`equalsIgnoreCase, toLowerCase, toUpperCase`

Manipulate String with For-each Loop

```
public static string replaceLetter(String word, char gone, char new1)
{
        char[] cArray = word.toCharArray();
        for (char c: cArray) {
                if (c == gone) {
                        c = new1;
                }
        }
        return word;
        return new String(cArray);
}
```

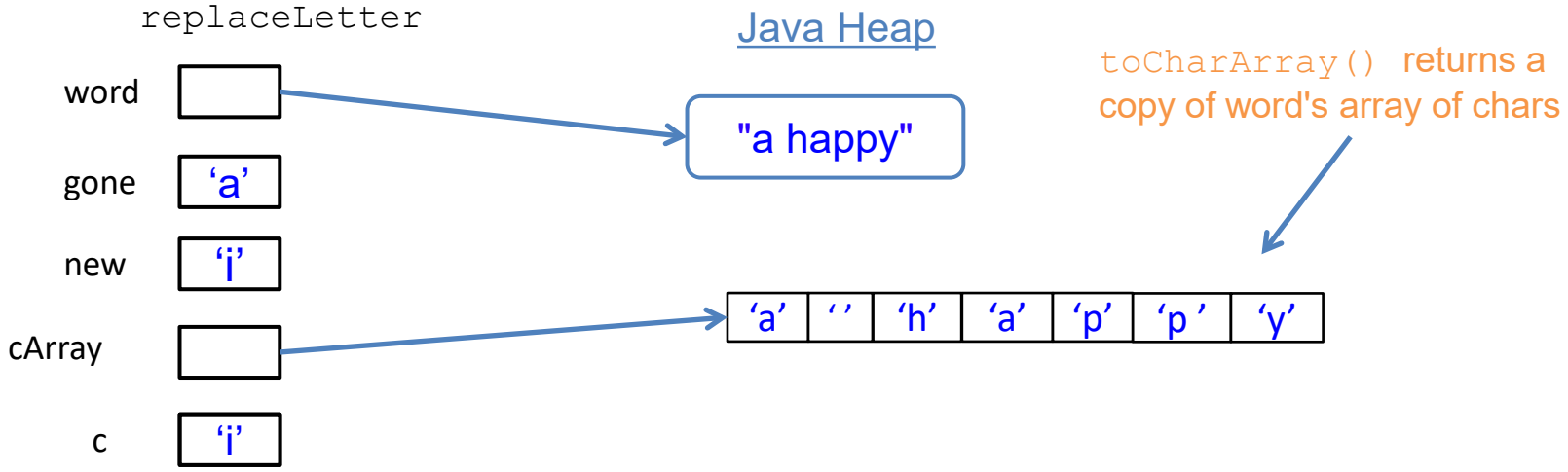
Does this method successfully return a modified word?

Attempt #1: NO

Attempt #2: NO

Let's trace the code with memory model diagram

```
// replaceLetter("a happy", 'a', 'i') -> "i hippy"??
```



Manipulate String with For-each Loop (Contd.)

```
public static string replaceLetter(String word, char goneIn, char newIn)
{
        char[] cArray = word.toCharArray();
        char[] cArrayMod = new char[cArray.length];
        int i = 0;
        for (char c: cArray) {
                if (c == goneIn)
                        cArrayMod[i] = newIn;
                else
                        cArrayMod[i] = c;
                i++;
        }
        return new String(cArrayMod);
}
```

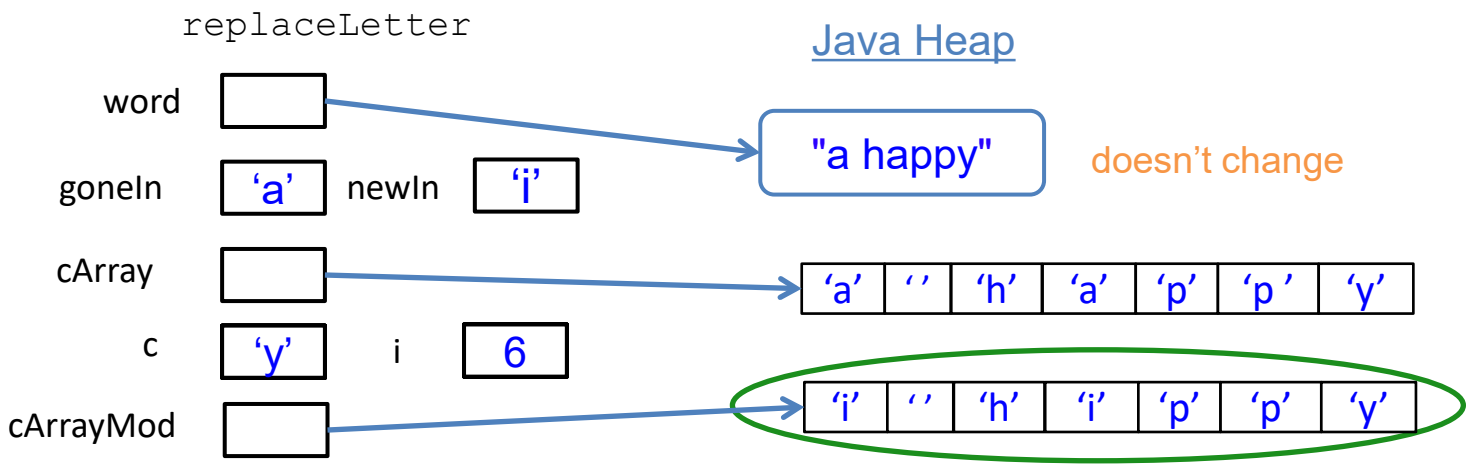
Does this method successfully return a modified word?

Attempt #1: NO

Attempt #2: NO

Attempt #3: YES

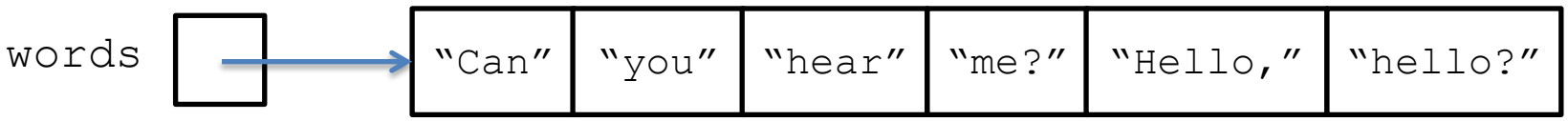
// replaceLetter("a happy", 'a', 'i') -> "i hippy"??



Count the number of words

Use String method `split(String pattern)` to split apart the String.

```
String text = "Can you hear me? Hello, hello?";
String[] words = text.split(" ");
```



what if we add an extra space here

```
String text = "Can you hear me?  Hello, hello?";
String[] words = text.split(" ");
```

split

`public String[] split(String regex)` ← it doesn't take a string

Splits this string around matches of the given regular expression.

Introduction to Regular Expressions (Regex)

Regular expression's basic units are characters, and it represents the pattern we are trying to match.

```
String text = "Hello  hello?";
String[] words = text.split(" ");
```

"Hello"	""	"hello?"

This single space is a regular expression. It matches single spaces

3 ways to combine

Repetition | Concatenation | Alternation

Repetition: + means 1 or more

```
String text = "Hello  hello?";
String[] words = text.split(" +");
```

"Hello"	"hello?"

Matches 1 or more spaces in a row

Create More Complicated Regex

```
public class Document {
        private String text; // The text of the whole document
        protected List<String> getTokens(String pattern)
}
```

returns a List of "tokens"

regex defining the "tokens"

Assume you have a Document object, d, whose text is "Hello hello?"

```
d.getTokens(" +");
-> [" "]
```

Matches 1 or more spaces

Repetition

Assume you have a Document object, d, whose text is "Splitting a string, it's as easy as 1 2 33! Right?"

```
d.getTokens("it");
-> ["it", "it"]
```

Two regular expressions side by side. Matches when both appear one after the other

Concatenation

Assume you have a Document object, d, whose text is "Splitting a string, it's as easy as 1 2 33! Right?"

```
d.getTokens("it+");
-> ["itt", "it"]
```

+ means "one or more"

Concatenation and Repetition

Create More Complicated Regex (Contd.)

```
public class Document {
        private String text; // The text of the whole document
        protected List<String> getTokens(String pattern)
}
```

Assume you have a Document object, d, whose text is "Splitting a string, it's as easy as 1 2 33! Right?"

```
d.getTokens("i(t+)");
-> ["itt", "it"]
```

Use parens to group r.e.'s if you are not sure of grouping

Concatenation and Repetition

Assume you have a Document object, d, whose text is "Splitting a string, it's as easy as 1 2 33! Right?"

```
d.getTokens("it*");
-> ["itt", "i", "i", "it", "i"]
```

* means "zero or more"

Assume you have a Document object, d, whose text is "Splitting a string, it's as easy as 1 2 33! Right?"

```
d.getTokens("it|st");
-> ["it", "st", "it"]
```

| means OR

Alternation

Create More Complicated Regex (Contd.)

```
public class Document {
        private String text; // The text of the whole document
        protected List<String> getTokens(String pattern)
}
```

Assume you have a Document object, d, whose text is "Splitting a string, it's as easy as 1 2 33! Right?"

```
d.getTokens("[123]");
-> ["1", "2" , "3", "3"]
```

[ ] mean match "anything in the set"

Character classes

```
d.getTokens("[1-3]");
-> ["1", "2" , "3", "3"]
```

- indicates a range
(any character between 1 and 3)

Assume you have a Document object, d, whose text is "Splitting a string, it's as easy as 1 2 33! Right?"

```
d.getTokens("[a-f]");
-> ["a", "a", "e", "a", "a"]
```

- indicates a range
(any character between a and f)

Assume you have a Document object, d, whose text is "Splitting a string, it's as easy as 1 2 33! Right?"

```
d.getTokens("[^a-z123 ]");
-> ["S", ",", "'", "!", "R", "?"]
```

^ indicates NOT any characters in this set

Negation

Excluding a character

Some Practices

```
public class Document {
        private String text; // The text of the whole document
        protected List<String> getTokens(String pattern)
}
```

Expression	Matches
"a*"	Zero or more a's
"a+"	1 or more a's
"[a-f]"	Any character between a and f
"[^a-cz]"	Any character which is not between a-c and not z
"[abc]+"	1 or more of the character a, b, or c in a row
"abc"	The characters abc in a row
"a|b"	The character a or the character b

Assume you have a Document object, d, whose text is "`Splitting a string, it's as easy as 1 2 33! Right?`"

```
d.getTokens("__________");
-> ["1", "2" , "33"]
```

Which of the following regular expressions can you insert in the blank so that it will give the output shown? Select all that apply.

A. "[1233]" ✗ -> ["1", "2", "3", "3"]

same as [123]

B. "[1,2,33]" ✗ -> [",", "1", "2", "3", "3"]

simply add comma to the group of letters that we're looking for.

C. "[0-9]+" ✓ -> ["1", "2" , "33"]

D. "[1-3]*" ✗ -> ["","","","","","","","","","","","","","","","",
,"","","","","","","","","","","","","","","","",
,"","","","","1","","2","","33","","","","",""," ","","", ""]

return empty string if the char is not in the group

E. "1|2|33" ✓ -> ["1", "2" , "33"]

Option C is FAR more versatile. It captures ANY non-negative integer (not just 1, 2, and 33).

Use Regex to Calculate Flesch Score

```
public class Document {
        private String text; // The text of the whole document
        protected List<String> getTokens(String pattern)
        public abstract int getNumWords();
        public abstract int getNumSentences();
```

given helper method

```
public class BasicDocument extends Document {
        @Override
        public int getNumWords() {
                List<String> tokens = getTokens("____________");
                return tokens.size();
        }
        @Override
        public int getNumSentences()
        {
                List<String> tokens = getTokens("____________");
                return tokens.size();
        }
```

Need a regex that matches "any word"

What constitutes a word?

"Any contiguous sequence of alphabetic characters"

What constitutes a sentence?

"A contiguous sequence of characters that does NOT include end of sentence punctuation."

"A sequence of any characters ending with end of sentence punctuation (. ! ?)"

http://www.tutorialspoint.com/java/java_regular_expressions.htm

Regex Exercises

- ^re\w+ed$
  - Matches strings that start with "re" and end with "ed" (like "received" or "renewed")
- ^re*ed$
  - Matches strings that start with "re", with e repeated 0 or more times, and end with "ed" (like "reed" or "reeed" or "reeeeeeeed")
- ^(re)*ed$
  - ed, reed, rereed, rererereered
- ^[re]*ed$
  - ed, eed, red, rrrred, eerreerred, rerereed
- ^[re]+ed$
  - eed, red, rrrred, eerreerred, rerereed, but NOT ed
- ^re{2}ed$
  - Matches reeed
- ^(re){2}ed$
  - Matches rereed
- re\wed
  - \w Matches any single word character (letter, digit, or underscore)
  - Matches "re" followed by exactly one word character, followed by "ed“ (like rexed, re1ed, re_ed, reAed)
- re.ed
  - . Matches any single character (except newline)
  - Matches “re” followed by exactly one single character, followed by “ed“ (like rexed, re-ed, re ed (including a space), re3ed, re.ed (matching a literal period)(

[re]*, [re]+

1. This regular expression will match: Any sequence of 'r' and 'e' characters, including an empty string
2. The characters 'r' and 'e' can appear in any order and any number of times

- Examples of Matching Strings
  - "" (empty string)
  - "r"
  - "e"
  - "re"
  - "er"
  - "ree"
  - "rre"
  - "eerr“
  - "rererere"
- Examples of Non-Matching Strings
  - "a" (contains a character other than 'r' or 'e')
  - "read" (contains characters other than 'r' or 'e')
  - "RED" (case-sensitive, uppercase letters don't match)
- [re]+ will match any sequence of 'r' and 'e' characters, will not match "" (empty string)

[^re]

- The ^ inside the square brackets [] negates the character class, meaning it matches any character except those listed.
- Components of the Regular Expression
  - [^re] - A negated character class that matches any single character that is NOT 'r' or 'e'
  - + - Quantifier that matches one or more occurrences of the preceding pattern
  - Matching Pattern
  - This regular expression will match:
  - One or more characters that are neither 'r' nor 'e'
  - Any sequence of characters as long as it doesn't contain 'r' or 'e'
- Examples of Matching Strings
  - "a"
  - "abc"
  - "123"
  - "xyz"
  - "!@#"
  - "The quick brown fox"
- Examples of Non-Matching Strings
  - "" (empty string, doesn't match because + requires at least one character)
  - "r" (contains 'r')
  - "e" (contains 'e')
  - "read" (contains both 'r' and 'e')