CS 106A, Lecture 9
String and char

reading:

Art & Science of Java, 8.1 - 8.2

Lecture at a glance

- Today we will learn about **strings**.
  - A string is a sequence of characters.
  - Strings can be manipulated or read as input into a program.
  - Lots of interesting data is stored as strings: names, addresses, books, songs, poems, DNA, ...

- We will also learn about the **`char`** data type.
  - A `char` is a single character of text.
  - A string is made up of 0-to-many `char` values.

Strings

- **string**: An object storing a sequence of text characters.

```
String name = "text";
String name = expression;
```

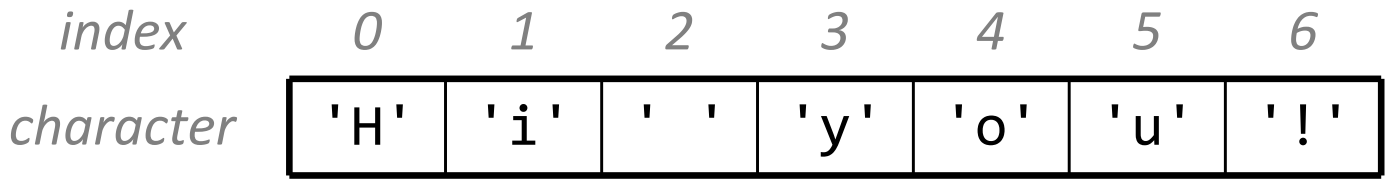
  - Examples:

```
String name = "Sean Combs";

int x = 3;
int y = 5;
String point = "(" + x + ", " + y + ")";
```

Indexes

- Characters of a string are numbered with 0-based *indexes*:

```
String text = "Hi you!";
```



  - First character's index : 0
  - Last character's index : 1 less than the string's length

  - The individual characters are values of primitive type `char`

String methods

Method name	Description
`s.indexOf(str)`	index where the start of the given string appears in this string (-1 if not found)
`s.length()`	number of characters in this string
`s.substring(index1, index2)` or `s.substring(index1)`	the characters in this string from *index1* (inclusive) to *index2* (exclusive); if *index2* is omitted, grabs till end
`s.toLowerCase()`	a new string with all lowercase letters
`s.toUpperCase()`	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String className = "CS 106A yay!";
println(className.length());   // 12
```

Method examples


```
// index      012345678901234567890123456
String s1 = "Snoop Dogg";
String s2 = "Marshall 'Eminem' Mathers";

println(s1.length());           // 10
println(s1.indexOf("o"));       // 2
println(s1.substring(6, 9));    // "Dog"

String s3 = s2.substring(18);   // "Mathers"
println(s3.toLowerCase());      // mathers
```

- Given the following string:

```
// index        01234567890123456789
String book = "Art & Science of Java";
```

  - How would you extract the word "Science" ?

Modifying a string

- Methods like `substring` and `toLowerCase` build and *return* a new string, rather than modifying the current string.

```
String s = "lil bow wow";
s.toUpperCase();
println(s);   // lil bow wow
```

- To modify a variable's value, you must reassign it:

```
String s = "lil bow wow";
s = s.toUpperCase();
println(s);   // LIL BOW WOW
```

String as user input

- The `readLine` method of a `ConsoleProgram` reads a line of input and returns it as a `String`.

```
String word = readLine("What's your favorite word? ");
println(word + " has " + word.length()
        + " characters and starts with "
        + word.substring(0, 1));
```

  - Output:

```
What's your name? defenestration
defenestration has 14 characters and starts with d
```

Name game exercise

- Write a console program that outputs "The Name Game", printing the following rhyme about the person's first and last name.
  - You may assume that the user types a string with exactly one space.

```
What is your name? Fifty Cent

Fifty Fifty, bo-Bifty
Banana-fana fo-Fifty
Fee-fi-mo-Mifty
FIFTY!

Cent, Cent, bo-Bent
Banana-fana fo-Fent
Fee-fi-mo-Ment
CENT!
```

Name game solution

```
import acm.program.*;

public class NameGame extends ConsoleProgram {
    public void run() {
        String name = readLine("What is your name? ");
        int spaceIndex = name.indexOf(" ");
        String firstName = name.substring(0, spaceIndex);
        String lastName = name.substring(spaceIndex + 1);
        song(firstName);
        song(lastName);
    }

    public void song(String name) {
        String rest = name.substring(1);
        println();
        println(name + ", " + name + ", bo-B" + rest);
        println("Banana-fana fo-F" + rest);
        println("Fee-fi-mo-M" + rest);
        println(name.toUpperCase() + "!");
    }
}
```

Cumulative string

- **cumulative string:** Creating a temporary string that starts out empty but grows over time. (Similar to "cumulative sum" code.)

```
String s = "";
s += "H";    // s is "H"
s += "e";    // s is "He"
s += "y";    // s is "Hey"
```

  - More commonly the string is accumulated in a loop:

```
String s = "";
for (int i = 0; i < 5; i++) {
    s += i;
}            // s is "01234"
```

Exercises w/ return

stutter
reverse

- Write a method named **`stutter`** that accepts a string parameter and returns that string with two consecutive copies of each character.
  - `stutter("Hello!")` returns `"HHeellllloo!!"`

- Write a method named **`reverse`** that accepts a string parameter and returns that string with its characters in the opposite order.
  - `reverse("Hi There!")` returns `"!erehT iH"`

  - Note that these methods *return* the new string, not just print it. (Why? What's the difference?)

String solutions

```
// These methods accumulate a string and return it.
public String stutter(String s) {
    String result = "";
    for (int i = 0; i < s.length(); i++) {
        String character = s.substring(i, i + 1);
        result += character;
        result += character;
    }
    return result;
}

public String reverse(String s) {
    String result = "";
    for (int i = s.length() - 1; i >= 0; i--) {
        result += s.substring(i, i + 1);
    }
    return result;
}
```

Comparing strings

```
String name = readLine("What is your name? ");
if (name == "Coolio") {
    println("Been spendin' most our lives");
    println("Living in a gangsta's paradise");
}
```

- The code above will compile, but it will not print the text.
- The == and != operators compare objects by *reference* (seen later), so it often gives `false` even when two strings have the same letters.

- Instead, strings are compared using a method named **`equals`**.

```
if (name.equals("Coolio")) { ...
```

String test methods

Method	Description
`s.equals(str)`	whether two strings contain the same characters
`s.equalsIgnoreCase(str)`	whether two strings contain the same characters, ignoring upper vs. lower case
`s.startsWith(str)`	whether one contains other's characters at start
`s.endsWith(str)`	whether one contains other's characters at end
`s.contains(str)`	whether the given string is found within this one

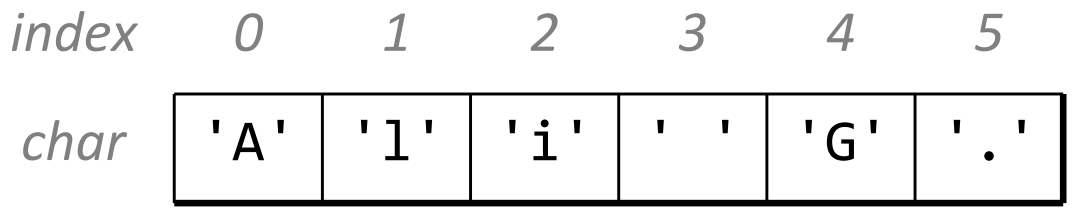
```
String name = readLine("What is your name? ");
if (name.startsWith("Prof")) {
    println("When are your office hours?");
} else if (name.equalsIgnoreCase("MARTY")) {
    println("Never mind.");
}
```

Type char

Type char

- **char** : A primitive type representing single characters.
  - A `String` is stored internally as an array of `char`

```
String s = "Ali G.";
```



  - It is legal to have variables, parameters, returns of type `char`
    - surrounded with apostrophes: `'a'` or `'4'` or `'\n'` or `'\''`

```
char letter = 'P';
println(letter);                // P
println(letter + " Diddy");     // P Diddy
```

The charAt method

- The chars in a String can be accessed using the charAt method.
  - accepts an int index parameter and returns the char at that index
  - similar to, but not quite the same as, `s.substring(i, i+1)`

```
String food = "cookie";
char first = food.charAt(0);            // 'c'
println(first + " is for " + food);     // c is for cookie
```

- You can use a for loop to print or examine each character.

```
String course = "106A";
for (int i = 0; i < course.length(); i++) {    // 1
    char c = course.charAt(i);                 // 0
    println(c);                                // 6
}                                              // A
```

Comparing char values

- Unlike with `Strings`, you *can* compare `chars` with ==, !=, etc.

```
String word = readLine("Type a word: ");
char last = word.charAt(word.length() - 1);
if (last == 's') {
    println(word + " is plural.");
}
```

  - You can even loop over a range of character values:

```
// prints the alphabet
for (char c = 'a'; c <= 'z'; c++) {
    print(c);
}
```

char and int

- Each `char` is mapped to an integer value internally
  - Called an **ASCII value**

```
'A' is 65        'B' is 66        ' ' is 32
'a' is 97        'b' is 98        '*' is 42
```

  - Mixing `char` and `int` causes automatic conversion to `int`

    `'a' + 10` is 107, `'A' + 'A'` is 130

  - To convert an `int` into the equivalent `char`, *type-cast* it with `()`.

    `(char) ('a' + 2)` is `'c'`

char and String

- "h" is a String, but 'h' is a char (they are different)

- A String is an object; it contains methods.

```
String s = "h";
s = s.toUpperCase();                // "H"
int len = s.length();               //  1
char first = s.charAt(0);           // 'H'
```

- A char is primitive; you can't call methods on it.

```
char c = 'h';
c = c.toUpperCase();                // ERROR
s = s.charAt(0).toUpperCase();      // ERROR
```

  - What is s + 1 ? What is c + 1 ?
  - What is s + s ? What is c + c ?

String/char exercise

- **Q:** What is the result of the following code?

```
String s = "dracula";
for (int i = 0; i < s.length(); i++) {
    char first = s.charAt(i);
    char last  = s.charAt(s.length() - 1 - i);
    String middle = s.substring(i, s.length() - 1 - i);
    s = last + middle + first;
}
```

  - **A.** The string `s` is reversed to become `"alucard"`.
  - **B.** The string `s` is mirrored to become `"draculaalucard"`.
  - **C.** The string `s` remains the same, `"dracula"`.
  - **D.** none of the above

Caesar cipher

CaesarCipher

- The *Caesar cipher* or *rotation cipher* is a crude system of encoding strings by shifting every letter forward by a given number.
- Write a program that encodes/decodes using a Caesar cipher.

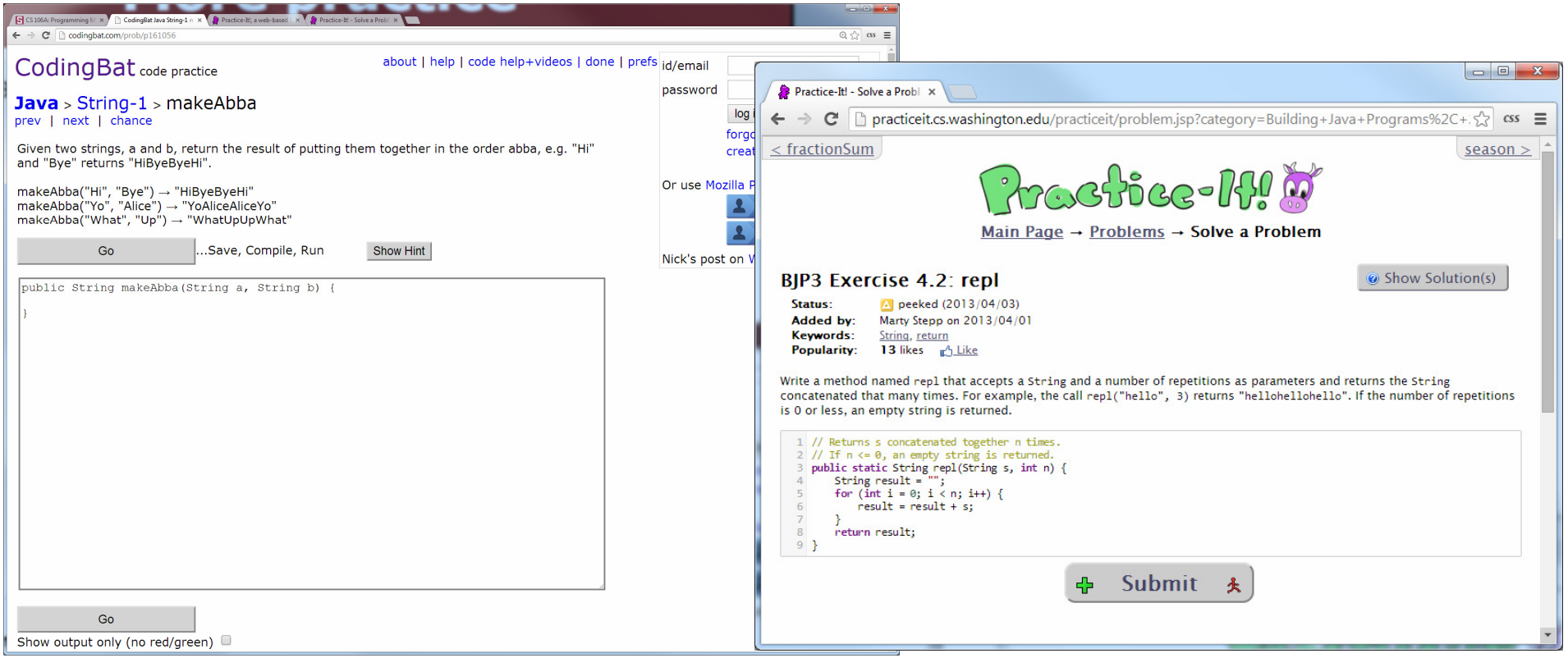
```
Your message? Attack zerg at dawn
Encoding key? 3
DWWDFN CHUJ DW GDZQ

Your message? dwwdfn chuj dw gdzq
Encoding key? -3
ATTACK ZERG AT DAWN
```

Cipher solution

```
public class CaesarCipher extends ConsoleProgram {
    public void run() {
        String message = readLine("Your message? ");
        int key = readInt("Encoding key? ");
        println(shift(message, key));
    }
    public String shift(String message, int amount) {
        message = message.toUpperCase();
        String result = "";
        for (int i = 0; i < message.length(); i++) {
            char c = message.charAt(i);
            if (c >= 'A' && c <= 'Z') {
                c += amount;
                if (c > 'Z') {
                    c -= 26;
                } else if (c < 'A') {
                    c += 26;
                }
            }
            result += c;
        }
        return result;
    }
}
```

More practice

- Stanford students can access these web sites to practice methods:
  - CodeStepByStep: http://codestepbystep.com/
  - Practice-It: http://practiceit.cs.washington.edu/
  - CodingBat: http://codingbat.com/



Overflow (extra) slides

String exercises

isVowel
isAllVowels

- Write a method named **`isVowel`** that returns whether a `String` is a vowel (a, e, i, o, or u), case-insensitively.
  - `isVowel("q")` returns `false`
  - `isVowel("A")` returns `true`
  - `isVowel("e")` returns `true`

- Write a method named **`isAllVowels`** that returns `true` only if every character in a `String` is a vowel.
  - `isAllVowels("eIeIo")` returns `true`
  - `isAllVowels("oink")` returns `false`

String solutions

```
public boolean isVowel(String s) {
    return s.equalsIgnoreCase("a") || s.equalsIgnoreCase("e") ||
           s.equalsIgnoreCase("i") || s.equalsIgnoreCase("o") ||
           s.equalsIgnoreCase("u");
}


public boolean isAllVowels(String s) {
    for (int i = 0; i < s.length(); i++) {
        String letter = s.substring(i, i + 1);
        if (!isVowel(letter)) {
            return false;
        }
    }
    return true;
}
```

Character methods

Method	Description
`Character.isDigit(`***ch***`)`	true if *ch* is `'0'` through `'9'`
`Character.isLetter(`***ch***`)`	true if *ch* is `'a'` through `'z'` or `'A'` through `'Z'`
`Character.isLowerCase(`***ch***`)`	true if *ch* is `'a'` through `'z'`
`Character.isUpperCase(`***ch***`)`	true if *ch* is `'A'` through `'Z'`
`Character.isWhitespace(`***ch***`)`	true if *ch* is a space, tab, new line, etc.
`Character.toLowerCase(`***ch***`)`	returns lowercase equivalent of a letter
`Character.toUpperCase(`***ch***`)`	returns uppercase equivalent of a letter

```
String s2 = "";
for (int i = 0; i < s.length(); i++) {
    char c = Character.toUpperCase(s.charAt(i));
    s2 += c;
    s2 += c;
}
```