

# CSC 221: Computer Programming I

Fall 2004

## String manipulation

- Java Strings: declaration, assignment, printing, concatenation
- String traversal, construction, `Character.toLowerCase`
- String methods: `length`, `charAt`, `indexOf`, `substring`
- applications: Pig Latin translator
- Strings vs. primitives: `equals`, `compareTo`
- unit testing

1

## Java strings

recall: String is a Java class that is automatically defined for you

- a String object encapsulates a sequence of characters
- you can declare a String variable and assign it a value just like any other type

```
String firstName = "Dave";
```

- you can display Strings using `System.out.print` and `System.out.println`

```
System.out.println(firstName);
```

- the '+' operator concatenates two strings (or string and number) together

```
String str = "foo" + "lish";  
str = str + "ly";
```

```
int age = 19;  
System.out.println("Next year, you will be " + (age+1));
```

2

## String methods

in addition, there are many useful methods defined for Strings

```
int length()
```

returns the length of the String str

```
String str = "foobar";  
System.out.println( str.length() );  
System.out.println( str.charAt(0) );  
System.out.println( str.charAt(1) );  
System.out.println( str.charAt(str.length()-1) );
```

```
char charAt(int index)
```

returns the character at specified index

- first index is 0
- last index is str.length()-1

```
String str = "foobar";  
for (int i = 0; i < str.length(); i++) {  
    System.out.print(str.charAt(i));  
}
```

*if index < 0 || index >= str.length(),  
an error occurs*

```
String str = "foobar";  
for (int i = str.length()-1; i >= 0; i--) {  
    System.out.print(str.charAt(i));  
}
```

3

## Traversing & constructing Strings

since the length of a String can be determined using the `length` method, a for loop can be used to traverse the String

- as you access individual characters, can test and act upon values
- can even construct a new string out of individual characters

```
String str = "zaboomofoo";  
  
int count = 0;  
for (int i = 0; i < str.length(); i++) {  
    if (str.charAt(i) == 'o') {  
        count++;  
    }  
}
```

```
String copy = "";  
for (int i = 0; i < str.length(); i++) {  
    copy = copy + str.charAt(i);  
}
```

```
String copy = "";  
for (int i = 0; i < str.length(); i++) {  
    copy = str.charAt(i) + copy;  
}
```

4

## String utilities

we can define and encapsulate additional string operations in a class

- `StringUtils` will not have any fields, it simply encapsulates methods
- can define methods to be *static* – static methods can be called directly on the class

```
public class StringUtils
{
    /**
     * Reverses a string.
     * @param str the string to be reversed
     * @return a copy of str with the order of the characters reversed
     */
    public static String reverse(String str)
    {
        String copy = "";
        for (int i = 0; i < str.length(); i++) {
            copy = str.charAt(i) + copy;
        }
        return copy;
    }
    .
    .
    .
}
```

5

## Stripping a string

consider the task of removing spaces from a string

- need to traverse the string and check each char to see if it is a space
- if it is not, add that char to the copy string
- if it is a space?

```
/**
 * Strips all spaces out of a string.
 * @param str the string to be stripped
 * @return a copy of str with each space removed
 */
public static String strip(String str)
{
    String copy = "";
    for (int i = 0; i < str.length(); i++) {
        if (str.charAt(i) != ' ') {
            copy += str.charAt(i);
        }
    }
    return copy;
}
```

6

## Censoring a string

consider the task of censoring a word, i.e., replacing each vowel with '\*'

- need to traverse the string and check each char to see if it is a vowel
- if it is a vowel, add '\*' to the copy string
- if it is not, add the char to the copy string

```
/**
 * Censors a string by replacing all vowels with asterisks.
 * @param str the string to be censored
 * @return a copy of str with each vowel replaced by an asterisk
 */
public static String censor(String str)
{
    String copy = "";
    for (int i = 0; i < str.length(); i++) {
        if (isVowel(str.charAt(i))) {
            copy += '*';
        }
        else {
            copy += str.charAt(i);
        }
    }
    return copy;
}

/**
 * Determines if a character is a vowel (either upper or lower case).
 * @param ch the character to be tested
 * @return true if ch is a vowel, else false
 */
private static boolean isVowel(char ch)
{
    ?????????
}
```

7

## Testing for a vowel

a brute force approach would be to test every vowel separately

TEDIOUS!

note that `isVowel` is a private method

- it isn't really a general purpose String utility
- its purpose is to be used by `censor` – user doesn't need to know about it

```
private static boolean isVowel(char ch)
{
    if (ch == 'a') {
        return true;
    }
    else if (ch == 'A') {
        return true;
    }
    else if (ch == 'e') {
        return true;
    }
    else if (ch == 'E') {
        return true;
    }
    .
    .
    .
    else if (ch == 'u') {
        return true;
    }
    else if (ch == 'U') {
        return true;
    }
    else {
        return false;
    }
}
```

8

## Testing for a vowel (cont.)

we could simplify the code using the `Character.toLowerCase` method

- `toLowerCase` is a static method of the `Character` class
- it takes a character as input, and returns the lower case equivalent
- if the input is not a letter, then it simply returns it unchanged

```
private static boolean isVowel(char ch)
{
    ch = Character.toLowerCase(ch);

    if (ch == 'a') {
        return true;
    }
    else if (ch == 'e') {
        return true;
    }
    else if (ch == 'i') {
        return true;
    }
    else if (ch == 'o') {
        return true;
    }
    else if (ch == 'u') {
        return true;
    }
    else {
        return false;
    }
}
```

9

## Testing for a vowel (cont.)

could simplify using `||`

```
private boolean isVowel(char ch)
{
    ch = Character.toLowerCase(ch);

    if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
        return true;
    }
    else {
        return false;
    }
}
```

since the code returns the same value as the test, can avoid the if altogether

```
private boolean isVowel(char ch)
{
    ch = Character.toLowerCase(ch);

    return (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u');
}
```

boolean expressions involving `||` or `&&` are evaluated intelligently via *short-circuit evaluation*

- expressions are evaluated left to right
- can stop evaluating `||` expression as soon as a part evaluates to true  
→ entire expression is true
- can stop evaluating `&&` expression as soon as a part evaluates to false  
→ entire expression is false

10

## Testing for a vowel (cont.)

best solution involves the `String` method:

```
int indexOf(char ch)
int indexOf(String str)
```

returns the index where `ch/str` first appears in the string (-1 if not found)

for `isVowel`:

- create a `String` that contains all the vowels
- to test a character, call `indexOf` to find where it appears in the vowel `String`
- if return value `!= -1`, then it is a vowel

```
private boolean isVowel(char ch)
{
    String VOWELS = "aeiouAEIOU";

    return (VOWELS.indexOf(ch) != -1);
}
```

11

## Substring

the last `String` method we will consider is `substring`:

```
String substring(int start, int end)
```

returns the substring starting at index `start` and ending at index `end-1`

```
e.g.,   String str = "foobar";
        str.substring(0,3)           → "foo"
        str.substring(3, str.length()) → "bar"
```

```
/**
 * Capitalizes the first letter in the string.
 * @param str the string to be capitalized
 * @return a copy of str with the first letter capitalized
 */
public static String capitalize(String str)
{
    return Character.toUpperCase(str.charAt(0)) + str.substring(1, str.length());
}
```

12

## Pig Latin

suppose we want to translate a word into Pig Latin

- simplest version

nix → ixnay  
latin → atinlay

pig → igpay  
banana → ananabay

- to translate a word, move the last letter to the end and add "ay"

```
/**
 * Translates a string into Pig Latin
 * @param str the string to be converted
 * @return a copy of str translated into Pig Latin
 */
public static String pigLatin(String str)
{
    return str.substring(1, str.length()) + str.charAt(0) + "ay";
}
```

13

## opsoay?

using our method,

oops → opsoay

apple → ppleaay

for "real" Pig Latin, you must consider the first letter of the word

- if a consonant, then translate as before (move first letter to end then add "ay")
- if a vowel, simply add "way" to the end

oops → oopsway

apple → appleway

```
public static String pigLatin(String str)
{
    if (isVowel(str.charAt(0))) {
        return str + "way";
    }
    else {
        return str.substring(1, str.length()) + str.charAt(0) + "ay";
    }
}
```

14

## reightoncay?

using our method,

creighton → reightoncay

thrill → hrilltay

for "real" Pig Latin, if the word starts with a sequence of consonants,  
must move the entire sequence to the end then add "ay"

creighton → eightoncray

thrill → illthray

so, we need to be able to find the first occurrence of a vowel

HOW?

15

## Handling multiple consonants

```
/**
 * Finds the first occurrence of a vowel in a string.
 * @param str the string to be searched
 * @return the index where the first vowel in str occurs (-1 if no vowel)
 */
private static int findVowel(String str)
{
    for (int i = 0; i < str.length(); i++) {
        if (isVowel(str.charAt(i))) {
            return i;
        }
    }
    return -1;
}

public static String pigLatin(String str)
{
    int firstVowel = findVowel(str);

    if (firstVowel <= 0) {
        return str + "way";
    }
    else {
        return str.substring(firstVowel, str.length()) +
            str.substring(0, firstVowel) + "ay";
    }
}
```

16

## In-class exercise

modify `pigLatin` so that it preserves capitalization

computer → omputercay

science → iencesay

Creighton → Eightoncray

Nebraska → Ebraskanay

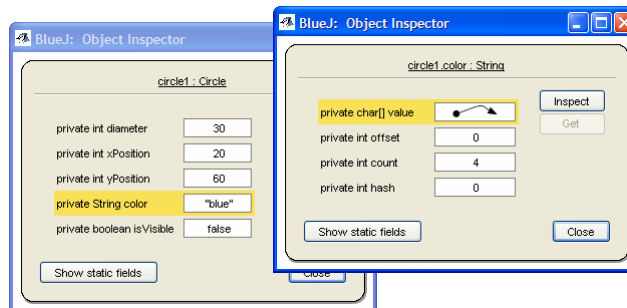
Omaha → Omahaway

17

## Strings vs. primitives

although they behave similarly to primitive types (int, double, char, boolean),  
Strings are different in nature

- String is a class that is defined in a separate library: `java.lang.String`  
→ a String value is really an object
- recall the distinction: you can call methods on a String, e.g., `str.length()`
- also, you can *Inspect* the String fields of an object



18

## Comparing strings

comparison operators (< <= > >=) are defined for primitives but not objects

```
String str1 = "foo", str2 = "bar";  
if (str1 < str2) ...           // ILLEGAL
```

== and != are defined for objects, but don't do what you think

```
if (str1 == str2) ...         // TESTS WHETHER THEY ARE THE  
                             // SAME OBJECT, NOT WHETHER THEY  
                             // HAVE THE SAME VALUE!
```

Strings are comparable using the equals and compareTo methods

```
if (str1.equals(str2)) ...   // true IF THEY REPRESENT THE  
                             // SAME STRING VALUE  
  
if (str1.compareTo(str2) < 0) ... // RETURNS -1 if str1 < str2  
                                 // RETURNS 0 if str1 == str2  
                                 // RETURNS 1 if str1 > str2
```

19

## Comparison example

suppose we wanted to compare two names to see which comes first alphabetically

- Kelly Jones < Kelly Miller < Chris Smith < Pat Smith

```
public static void compareNames(String myFirst, String myLast,  
                               String yourFirst, String yourLast)  
{  
    int lastCompare = myLast.compareTo(yourLast);  
    int firstCompare = myFirst.compareTo(yourFirst);  
  
    if (lastCompare < 0 || (lastCompare == 0 && firstCompare < 0)) {  
        System.out.println("My name comes before yours alphabetically!");  
    }  
    else if (lastCompare > 0 || (lastCompare == 0 && firstCompare > 0)) {  
        System.out.println("Your name comes before mine alphabetically!");  
    }  
    else {  
        System.out.println("We have the same name!");  
    }  
}
```

20

## String method summary

<code>int length()</code>	returns number of chars in String
<code>char charAt(int index)</code>	returns the character at the specified index (indices range from 0 to <code>str.length()-1</code> )
<code>int indexOf(char ch)</code> <code>int indexOf(String str)</code>	returns index where the specified char/substring first occurs in the String (-1 if not found)
<code>String substring(int start, int end)</code>	returns the substring from indices start to (end-1)
<code>String toUpperCase()</code> <code>String toLowerCase()</code>	returns copy of String with all letters uppercase returns copy of String with all letters lowercase
<code>bool equals(String other)</code> <code>int compareTo(String other)</code>	returns true if other String has same value returns -1 if less than other String, 0 if equal to other String, 1 if greater than other String

### ALSO, from the Character class:

<code>char Character.toLowerCase(char ch)</code>	returns lowercase copy of ch
<code>char Character.toUpperCase(char ch)</code>	returns uppercase copy of ch
<code>boolean Character.isLetter(char ch)</code>	returns true if ch is a letter
<code>boolean Character.isLowerCase(char ch)</code>	returns true if lowercase letter
<code>boolean Character.isUpperCase(char ch)</code>	returns true if uppercase letter <sup>21</sup>

## Testing code

when you design and write code, how do you know if it works?

- run it a few times and assume it's OK?

to be convinced that code runs correctly in all cases, you must analyze the code and identify special cases that are handled

- then, define a test data set (inputs & corresponding outputs) that covers those cases
- e.g., for Pig Latin,
  - words that start with single consonant: "foo" → "oofay" "banana" → "ananabay"
  - words that start with multiple consonants: "thrill" → "illthray" "cheese" → "eesechay"
  - words that start with vowel: "apple" → "appleway" "oops" → "oopsway"
  - words with no vowels: "nth" → "nthway"
  - words that are capitalized: "Creighton" → "Eightoncray" "Omaha" → "Omahaway"

## Unit testing in BlueJ

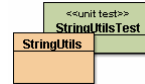
*unit testing*: test each component (e.g., method) of a project independently

- once the components are individually tested, their combined behavior must be tested

IMPORTANT: when you make any changes to existing code, you must rerun all of its test data set

BlueJ provides a mechanism for automatically performing unit testing

- right-click on a class and select "Create Test Class"
- this creates a separate class named "CLASS Test"
- specify test data by right-clicking on Test class and selecting "Create Test Method"
- after inputting a name (e.g., "SingleConsonant"), perform the desired steps (e.g., make method call & verify result), then click "End"
- can create numerous test methods to cover the entire test data set
- can run all tests in sequence simply by clicking on the "Run tests" button



23

## Palindrome

suppose we want to define a method to test whether a word is a palindrome (i.e., reads the same forwards and backwards)

```
isPalindrome("bob") → true           isPalindrome("madam") → true
isPalindrome("blob") → false         isPalindrome("madame") → false
```

download `StringUtils.java` and `StringUtilsTest.java`

define `isPalindrome` method returns true if the word is a palindrome

extend your `isPalindrome` method to handle phrases

- need to ignore spaces, punctuation & capitalization

```
isPalindrome("Madam, I'm Adam.") → true
isPalindrome("Able was I ere I saw Elba.") → true
isPalindrome("A man, a plan, a canal: Panama.") → true
```

24