1
Lecture 14 - Strings

topics:
Lab 7 is due today.
   review test 3 solutions
   discuss Lab 8
ch. 10 – Text Processing;
Wrapper classes
String class
constructing strings
string methods: length, charAt, getChars
comparing strings
substrings
StringBuffer

(review from ch. 2)
1. Strings are objects in Java, based on the String class (java.lang package)

To create (instantiate) a String, call one of the String class constructors

String sName = new String(“mickey mouse”); // sName is a variable reference to a String object

Or use the shorthand:

String sCity = “San Francisco”; // the result is the same

A. String methods
   length() – returns the # of characters in the string, which is immutable(unchangeable)
   length() is a method, not a variable, as in array.length.

   “hello”.length() -> 5
   sCity.length() -> 13

   charAt() – returns the character at an integer argument. the first character is at index 0
   sCity.charAt(3) -> ‘ ‘
   sCity.charAt(0) -> ‘S’
getChars() – used to copy characters from a string into character array

syntax: `stringRef.getChars(startIndex, pastIndex, arrayName, writeIndex)`

`sCity.getChars(0, 3, charArray, 0);` // copies ‘S’, ‘a’, and ‘n’ into the first 3 elements of charArray.

B. Strings methods for comparing

The computer compares the Unicode values of each corresponding character pair when comparing 2 strings

“San Francisco” comes before
“San Jose”

equals() – compares the String data (the characters) and returns true or false

// shortcut syntax:
String sOne = “Hello”;
String sTwo = “Hello”;

// full syntax:
String sOne = new String(“Hello”);
String sTwo = new String(“Hello”);

sOne.equals(sTwo) -> true because both contain the same characters

BUT the equality operator == compares strings to see if they are the same OBJECT, not if they have the same data.

// full syntax instantiates 2 different String objects, at different addresses
String sOne = new String(“Hello”);
String sTwo = new String(“Hello”);
(sOne == sTwo) -> false

BUT if we use the shortcut syntax, and assign the same literal string to 2 variables, Java creates just one object, so they are aliases, and sOne == sTwo → true!
String sOne = Hello”;
String sTwo = Hello”;

// also, when you copy on object into another, you are copying the reference:
String sOne = “Hello”;
String sTwo = sOne;
(sOne == sTwo) -> true

compareTo() returns 0 if strings are equal, a negative number if string we call the method on is less than string argument, and a positive number if the reverse.

We can also compare parts of strings, with method regionMatches() and make case-insensitive comparisons with equalsIgnoreCase()

C. Substring methods

a) methods which return the location of single characters or substrings in a larger string:

indexOf() – searches from the beginning of the String for a single char or string

str.indexOf(‘a’, 5 ) -> returns index of first ‘a’ starting at index 5, -1 if not found

lastIndexOf() - searches from the end of the String

str.lastIndexOf(‘abc’);
str.lastIndexOf(‘a’, 20); // searches backward from index 20

b) methods which return substrings

substring() – returns a substring from startIndex to the end or from startIndex until 1 character before endIndex

syntax: stringRef.substring(startIndex {, endIndex} ) // if no endIndex, returns all chars to the end

String sName = “Catherine”;
String sWord = sName.substring( 2, 5); // sWord gets “the”
String sText = sName.substring( 5 ); // sText gets “rine”

replacing a substring

str.replace(‘a’, ‘A’ ) // returns a new string where each ‘a’ is now ‘A’
if none found, returns the original string
concat method concatenates two strings and returns a new string object. The original strings are not changed.

String sOne = “Mickey”; 
String sTwo = “Mouse”; 
String sThree = sOne.concat(sTwo); // sThree gets “MickeyMouse”

D) method valueOf() converts the contents of an identifier to its string representation.

int number = 10; 
String sName = String.valueOf(number); // sName gets “10”

String.valueOf(true) -> “true”

original values : boolean, char, int, long, float, double, char array

E) conversions
toCharArray() – returns all the characters in a String as an array of chars:

char[] letterArray = sName. toCharArray();

letterArray looks like this:
    C a t h e r i n e

to upper case:
s1.toUpperCase() -> new string

to lower case:
s1.toLowerCase()

trims leading spaces:
s1.trim()
2. StringBuffer class

for dynamic strings – contents can be changed (not immutable)

A) constructors

new StringBuffer() //empty string with capacity of 16
new StringBuffer(10) // empty string with capacity of 10
new StringBuffer(“hello”) // sets capacity to 21 (5 in hello + 16) and contents to “hello”

B) methods

list on p. 642 – same set of methods as the String class

StringBuffer methods not found in String class:

    object.append(item) – there are many overloaded versions of this method

    toString() – returns String content

    object.insert(start, item)
    object.replace(start, end, str);// end is index after last char to replace

    StringBuffer strB = new StringBuffer(“ABCDEFGHIJ”);
    strB.replace(2, 5, “hello”); // now contains “ABhelloFGHIJ”

other methods for deleting substrings and chars, and replacing chars

3. StringTokenizer class

a token is is an item of data, represented as a substring, that is part of a larger string. Tokens are separated by white space or by delimiters, any specified character, such as a comma, dash, or colon. When we break out the separate substrings from a larger string into tokens, it is called tokenizing, and uses methods of the StringTokenizer class.

import java.util. StringTokenizer;
overloaded constructors:
StringTokenizer(String str); // uses white space as delimeter

```java
StringTokenizer str = new StringTokenizer("hello, how are you?");
while (str.hasMoreTokens())
    System.out.println(str.nextToken());
```
output:
hello,
how
are
you?

StringTokenizer(String str, String delimiters); // delimiters listed in string

```java
StringTokenizer str = new StringTokenizer("5,6,20,2", ",");
tokens are: 5
    6
    20
    2
```

StringTokenizer(String str, String delimiters, Boolean includeDelims);
// delimiters listed in string, true returns delimiters as tokens

```java
StringTokenizer str = new StringTokenizer("5,6,20,2", ",", true);
tokens are:
    5
    6
    20
    2
    ,
```
To extract tokens, use `hasMoreTokens()` to control while loop, or use `countTokens()` to control a for loop:

```java
StringTokenizer str = new StringTokenizer("hello, how are you?");
for (int j = str.countTokens(); j > 0 ; j--){
    System.out.println(str.nextToken());
}
```

Every time you get a token using `nextToken()`, it decreases the number of tokens left, so calling `countToken` again will return a smaller number.

Use the `trim()` method of the `String` class to trim leading or trailing white spaces in the `String` argument to the `StringTokenizer` constructor.

```java
String str = "    apple, pear,   banana   ";
StringTokenizer str = new StringTokenizer(str.trim(), ",");
```

without using `trim()`, the tokens would be:

"apple"
"pear"
"banana"

4. Wrapper classes – “wraps” around a primitive data type, such as a char, to provide useful methods for manipulating that type.
   Used for the static methods they provide, rather than to create useful objects.
   Wrapper class objects cannot be changed (like the `String` class) – they are immutable.
Any questions about the `Character` class methods?

Wrapper classes for numeric types:

In addition to the parse methods you have already used this semester, such as `Integer.parseInt()`, every class has a `toString()` method, which converts a number to its string equivalent. `toString` is a method which does not have to be explicitly called to be executed, for example:

```java
int myNum = 54
System.out.println(myNum); // this indirectly calls the Integer class toString() with myNum as the argument
```
we could have spelled it out:
System.out.println(Integer.toString(myNum));

or with a literal number:
System.out.println(54);
System.out.println(Integer.toString(54));

We won’t cover the other toString methods this semester, or the rest of sections 10.6 and 10.7