CS 113 – Computer Science I

Lecture 07 – String Methods & Recursion

Tuesday 09/26/2023
Announcements

• HW02:
  • Due tonight

• HW03 – releasing tonight
  • Due Monday 10/02

• Read & Follow Instructions
  • Don’t just skim the labs & homework
Agenda

String Comparison
Recursion
Comparing strings

• In Java, you cannot directly compare strings using ==

• Instead, use `compareTo`
  • Javadocs: https://docs.oracle.com/javase/7/docs/api/java/lang/String.html
compareTo

public int compareTo(String anotherString)

Compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The character sequence represented by this String object is compared lexicographically to the character sequence represented by the argument string. The result is a negative integer if this String object lexicographically precedes the argument string. The result is a positive integer if this String object lexicographically follows the argument string. The result is zero if the strings are equal; compareTo returns 0 exactly when the equals(Object) method would return true.

This is the definition of lexicographic ordering. If two strings are different, then either they have different characters at some index that is a valid index for both strings, or their lengths are different, or both. If they have different characters at one or more index positions, let k be the smallest such index; then the string whose character at position k has the smaller value, as determined by using the < operator, lexicographically precedes the other string. In this case, compareTo returns the difference of the two character values at position k in the two string -- that is, the value:

    this.charAt(k) - anotherString.charAt(k)

If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string. In this case, compareTo returns the difference of the lengths of the strings -- that is, the value:

    this.length() - anotherString.length()

Specified by:
    compareTo in interface Comparable<String>

Parameters:
    anotherString - the String to be compared.

Returns:
    the value 0 if the argument string is equal to this string; a value less than 0 if this string is lexicographically less than the string argument; and a value greater than 0 if this string is lexicographically greater than the string argument.
public int compareTo(String anotherString)

Parameters:

anotherString - the String to be compared.

Returns:

• the value 0 if the argument string is equal to this string;
• a value less than 0 if this string is lexicographically less than the string argument;
• and a value greater than 0 if this string is lexicographically greater than the string argument.
Comparing strings

• In Java, you cannot directly compare strings: use `compareTo`

String a = "apple";
String b = "banana";
if (a.compareTo(b) == 0) {
    System.out.println("a and b match!");
}
if (a.compareTo(b) != 0) {
    System.out.println("a and b DO NOT match!");
}
Lexicographic Values/Order

• Strings are **ordered lexicographically**
  
  • Generally, the same order as alphabetical order, with some caveats
  
  • The characters of a string each correspond to a number
<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>000</td>
<td>NUL (null)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
<td>SOH (start of heading)</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>002</td>
<td>STX (start of text)</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>003</td>
<td>ETX (end of text)</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>004</td>
<td>EOT (end of transmission)</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>005</td>
<td>ENQ (enquiry)</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>006</td>
<td>ACK (acknowledge)</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>007</td>
<td>BEL (bell)</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>010</td>
<td>BS (backspace)</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>011</td>
<td>TAB (horizontal tab)</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>012</td>
<td>LF (NL line feed, new line)</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>013</td>
<td>VT (vertical tab)</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>014</td>
<td>FF (NP form feed, new page)</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>015</td>
<td>CR (carriage return)</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>016</td>
<td>SO (shift out)</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>017</td>
<td>SI (shift in)</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>020</td>
<td>DLE (data link escape)</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>021</td>
<td>DC1 (device control 1)</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>022</td>
<td>DC2 (device control 2)</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>023</td>
<td>DC3 (device control 3)</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>024</td>
<td>DC4 (device control 4)</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>025</td>
<td>NAK (negative acknowledgment)</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>026</td>
<td>SYN (synchronous idle)</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>027</td>
<td>ETB (end of trans. block)</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>030</td>
<td>CAN (cancel)</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>031</td>
<td>EM (end of medium)</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>032</td>
<td>SUB (substitute)</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>033</td>
<td>ESC (escape)</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>034</td>
<td>FS (file separator)</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>035</td>
<td>GS (group separator)</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>036</td>
<td>RS (record separator)</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>037</td>
<td>US (unit separator)</td>
</tr>
</tbody>
</table>

Source: [www.LookupTables.com](https://www.asciitable.com/)
String first = "a";
String second = "A";
int asciia = (int) first.charAt(0);
int asciib = (int) second.charAt(0);
System.out.println("ASCII Code for "+first+" is " + asciia);
System.out.println("ASCII Code for "+second+" is " + asciib);

if (first.compareTo(second) == 0) {
    System.out.println(first+" is equal to "+second);
}
else if (first.compareTo(second) < 0) {
    System.out.println(first+" is less than "+second);
}
else if (first.compareTo(second) > 0) {
    System.out.println(first+" is greater than "+second);
}
Exercise: IsPrimary

Write a program that asks the user for a color and prints whether the color is primary or not.

• The primary colors are “red”, “green”, “blue”

• All other inputs are non-primary

$ java IsPrimary
Enter a color: green
green is not primary

$ java IsPrimary
Enter a color: blue
blue is primary
Top down design

1. Identify features of the program
   1. List them out!

2. Identify verbs and nouns in feature list
   1. Verbs: functions
   2. Nouns: objects/variables

3. Sketch major steps – how features should fit together
   1. Algorithm!

4. Write program skeleton
   1. Include method stubs (placeholders for our functions)
   2. method stub: empty function with parameters and return type

5. Implement and test method stubs one at a time
Smart way to wash dishes

Punt the problem to someone else

But we want to wash one dish so we can say we washed a dish
Recursion
Recursion

a function that calls itself

“Simple” way to solve “similar” problems
Creating a recursive algorithms

**Rule** that “does work” then ”calls itself” on a smaller version of the problem

**Base case** that handles the smallest problem

Prevents “infinite recursion”
Recursion example – print “hello” 5 times

**Rule:** Print “hello” once and then print “hello” 4 times

**Base case:** When the number of times to print is 0, stop printing
Recursive functions – base case

Conditional statement that prevents infinite repetitions

Usually handles cases where:

- input is empty
- problem is at its smallest size
Recursion Example - Factorial

\[ n! = n \times (n - 1) \times (n - 2) \times ... \times 1 \]

3! = 3 \times 2 \times 1 = 6

4! = 4 \times 3 \times 2 \times 1 = 24
Visualizing recursion — Factorial example

\[ \text{factorial}(5) = \]
\[ = 5 \times \text{factorial}(4) \]
\[ = 5 \times 4 \times \text{factorial}(3) \]
\[ = 5 \times 4 \times 3 \times \text{factorial}(2) \]
\[ = 5 \times 4 \times 3 \times 2 \times \text{factorial}(1) \]
\[ = 5 \times 4 \times 3 \times 2 \times 1 \]
Recursion Example – Contains letter

Write a method called “containsLetter” that determines if a String contains a given character

Question: What are the parameters?
   1. The String to be looking in
   2. The character to look for

Question: What is the return type?
Recursion Example – Contains letter

How can we break this problem down into smaller problems?

contains("l", "apple") =
contains("l", "a") OR
contains("l", "p") OR
contains("l", "p") OR
contains("l", "l") OR
contains("l", "e") OR
Recursion Visualization – Contains letter

contains("l", "apple") = 
contains("l", "apple")
    contains("l", "pple")
        contains("l", "ple")
            contains("l", "le")
                return true
Recursion Example – IndexOf letter

Write a method called IndexOf.

Arguments: String (haystack), Character (needle)

Return: the index of the character in the String, if the character isn't there, return:

-1.
Recursion Example – printVowels

Write a recursive function that prints just the vowels in a String
Recursion limitations

• Limited number of times we can recurse
  • Stackoverflow – too many frames

• Potentially memory inefficient
  • If we copy data in subproblems – we’ll worry about this in a few weeks

• Performance: might duplicate unnecessary work
  • We’ll define performance later in the semester
Style

• How we format our programs is *very* important
  • Like rules of etiquette around eating and keep a clean appearance
  • Like punctuation rules, it helps make text more readable

• Variable names should be descriptive

• Indentation is *very* important
  • Every statement inside a pair of braces must be indented

• Braces should be placed consistently