CS 113 – Computer Science I

Lecture 4 – Loops

Adam Poliak
09/13/2022
Announcements

• Assignment 00
  • Good job!
  • Grades coming out soon

• Assignment 01
  • Due Thursday 09/15

• Office hours:
  • Adam’s: 10:30-11:30am on Wednesdays
Agenda

• Announcements
• Recap
• String Comparison
• Loops
Comparing strings

• In Java, you cannot directly compare strings: use `compareTo`
  • Javadocs: https://docs.oracle.com/javase/7/docs/api/java/lang/String.html
**compareTo**

```java
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:

```java
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:

```java
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.
**compareTo**

```java
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:

\[
\text{this.charAt}(k) - \text{anotherString.charAt}(k)
\]

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:

\[
\text{this.length}() - \text{anotherString.length}()
\]

**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`

```java
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>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Char</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Char</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Char</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>000</td>
<td>NUL</td>
<td>(null)</td>
<td>32</td>
<td>20</td>
<td>040</td>
<td> 32;</td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>100</td>
<td> 64;</td>
<td>0</td>
<td>96</td>
<td>60</td>
<td>140</td>
<td> 96;</td>
<td>ˋ</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
<td>SOH</td>
<td>(start of heading)</td>
<td>33</td>
<td>21</td>
<td>041</td>
<td> 33;</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>101</td>
<td> 65;</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>141</td>
<td> 97;</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>002</td>
<td>STX</td>
<td>(start of text)</td>
<td>34</td>
<td>22</td>
<td>042</td>
<td> 34;</td>
<td>”</td>
<td>66</td>
<td>42</td>
<td>102</td>
<td> 66;</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>142</td>
<td> 98;</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>003</td>
<td>ETX</td>
<td>(end of text)</td>
<td>35</td>
<td>23</td>
<td>043</td>
<td> 35;</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>103</td>
<td> 67;</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>143</td>
<td> 99;</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>004</td>
<td>EOT</td>
<td>(end of transmission)</td>
<td>36</td>
<td>24</td>
<td>044</td>
<td> 36;</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>104</td>
<td> 68;</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>144</td>
<td> 100;</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>005</td>
<td>ENQ</td>
<td>(enquiry)</td>
<td>37</td>
<td>25</td>
<td>045</td>
<td> 37;</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>105</td>
<td> 69;</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>145</td>
<td> 101;</td>
<td>e</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>006</td>
<td>ACK</td>
<td>(acknowledge)</td>
<td>38</td>
<td>26</td>
<td>046</td>
<td> 38;</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>106</td>
<td> 70;</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>146</td>
<td> 102;</td>
<td>f</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>007</td>
<td>BEL</td>
<td>(bell)</td>
<td>39</td>
<td>27</td>
<td>047</td>
<td> 39;</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>107</td>
<td> 71;</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>147</td>
<td> 103;</td>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>010</td>
<td>BS</td>
<td>(backspace)</td>
<td>40</td>
<td>28</td>
<td>050</td>
<td> 40;</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>110</td>
<td> 72;</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>150</td>
<td> 104;</td>
<td>h</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>011</td>
<td>TAB</td>
<td>(vertical tab)</td>
<td>41</td>
<td>29</td>
<td>051</td>
<td> 41;</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>111</td>
<td> 73;</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>151</td>
<td> 105;</td>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>012</td>
<td>LF</td>
<td>(NL line feed, new line)</td>
<td>42</td>
<td>2A</td>
<td>052</td>
<td> 42;</td>
<td>)</td>
<td>74</td>
<td>4A</td>
<td>112</td>
<td> 74;</td>
<td>J</td>
<td>106</td>
<td>6A</td>
<td>152</td>
<td> 106;</td>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>013</td>
<td>VT</td>
<td>(vertical tab)</td>
<td>43</td>
<td>2B</td>
<td>053</td>
<td> 43;</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>113</td>
<td> 75;</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>153</td>
<td> 107;</td>
<td>k</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>014</td>
<td>FF</td>
<td>(NP form feed, new page)</td>
<td>44</td>
<td>2C</td>
<td>054</td>
<td> 44;</td>
<td></td>
<td>76</td>
<td>4C</td>
<td>114</td>
<td> 76;</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>154</td>
<td> 108;</td>
<td>l</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>015</td>
<td>CR</td>
<td>(carriage return)</td>
<td>45</td>
<td>2D</td>
<td>055</td>
<td> 45;</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>115</td>
<td> 77;</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>155</td>
<td> 109;</td>
<td>m</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>016</td>
<td>SO</td>
<td>(shift out)</td>
<td>46</td>
<td>2E</td>
<td>056</td>
<td> 46;</td>
<td></td>
<td>78</td>
<td>4E</td>
<td>116</td>
<td> 78;</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>156</td>
<td> 110;</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>017</td>
<td>SI</td>
<td>(shift in)</td>
<td>47</td>
<td>2F</td>
<td>057</td>
<td> 47;</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>117</td>
<td> 79;</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>157</td>
<td> 111;</td>
<td>o</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>020</td>
<td>DLE</td>
<td>(data link escape)</td>
<td>48</td>
<td>30</td>
<td>060</td>
<td> 48;</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>120</td>
<td> 80;</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>160</td>
<td> 112;</td>
<td>p</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>021</td>
<td>DC1</td>
<td>(device control 1)</td>
<td>49</td>
<td>31</td>
<td>061</td>
<td> 49;</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>121</td>
<td> 81;</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>161</td>
<td> 113;</td>
<td>q</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>022</td>
<td>DC2</td>
<td>(device control 2)</td>
<td>50</td>
<td>32</td>
<td>062</td>
<td> 50;</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>122</td>
<td> 82;</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>162</td>
<td> 114;</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>023</td>
<td>DC3</td>
<td>(device control 3)</td>
<td>51</td>
<td>33</td>
<td>063</td>
<td> 51;</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>123</td>
<td> 83;</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>163</td>
<td> 115;</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>024</td>
<td>DC4</td>
<td>(device control 4)</td>
<td>52</td>
<td>34</td>
<td>064</td>
<td> 52;</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>124</td>
<td> 84;</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>164</td>
<td> 116;</td>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>025</td>
<td>NAK</td>
<td>(negative acknowledge)</td>
<td>53</td>
<td>35</td>
<td>065</td>
<td> 53;</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>125</td>
<td> 85;</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>165</td>
<td> 117;</td>
<td>u</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>026</td>
<td>SYN</td>
<td>(synchronous idle)</td>
<td>54</td>
<td>36</td>
<td>066</td>
<td> 54;</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>126</td>
<td> 86;</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>166</td>
<td> 118;</td>
<td>v</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>027</td>
<td>ETB</td>
<td>(end of trans. block)</td>
<td>55</td>
<td>37</td>
<td>067</td>
<td> 55;</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>127</td>
<td> 87;</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>167</td>
<td> 119;</td>
<td>w</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>030</td>
<td>CAN</td>
<td>(cancel)</td>
<td>56</td>
<td>38</td>
<td>070</td>
<td> 56;</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>130</td>
<td> 88;</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>170</td>
<td> 120;</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>031</td>
<td>EM</td>
<td>(end of medium)</td>
<td>57</td>
<td>39</td>
<td>071</td>
<td> 57;</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>131</td>
<td> 89;</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>171</td>
<td> 121;</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>032</td>
<td>SUB</td>
<td>(substitute)</td>
<td>58</td>
<td>3A</td>
<td>072</td>
<td> 58;</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>132</td>
<td> 90;</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>172</td>
<td> 122;</td>
<td>z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>27</td>
<td>033</td>
<td>ESC</td>
<td>(escape)</td>
<td>59</td>
<td>3B</td>
<td>073</td>
<td> 59;</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>133</td>
<td> 91;</td>
<td></td>
<td>123</td>
<td>7B</td>
<td>173</td>
<td> 123;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 28  | 28 | 034 | FS   | (file separator) | 60 | 3C | 074 | &nbsp;60; | < | 92 | 5C | 134 | &nbsp;92; | | 124 | 7C | 174 | &nbsp;124; | |}

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”, “yellow”, “blue”

• All other inputs are non-primary
Agenda

- Announcements
- Recap
- String Comparison
- Loops
Exercise

Suppose we wanted to ask the user for 6 numbers (int) and output their sum?
Loops

• Easy way to repeat some computation

• Two kinds of loops:
  • While
  • For

• Loops repeat block of code until the condition becomes false
Example: While Loop

```java
int val = 0;
String valStr = "";
int sum = 0;

int count = 0;
while (count < 6) {
    System.out.print("Enter a number: ");
    valStr = System.console().readLine();
    val = Integer.parseInt(valStr);
    sum = sum + val;
    count = count + 1;
}
System.out.println("The sum is "+sum);
```
Tracing Loops

```c
int sum = 1;
int count = 0;
while (count < 3) {
    sum = sum + 2;
    count = count + 1;
}
```

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Count &lt; 6</th>
<th>count</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Tracing Loops

```c
int sum = 1;
int count = 0;
while (count < 3) {
    sum = sum + 2;
    count = count + 1;
}
```

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Count &lt; 6</th>
<th>count</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Tracing Loops

```c
int sum = 1;
int count = 0;
while (count < 3) {
    sum = sum + 2;
    count = count + 1;
}
```

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Count &lt; 6</th>
<th>count</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Tracing Loops

```c
int sum = 1;
int count = 0;
while (count < 3) {
    sum = sum + 2;
    count = count + 1;
}
```

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Count &lt; 6</th>
<th>count</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Iteration Count: 0

- 0: sum = 1, count = 0
- 1: sum = 3, count = 1
- 2: sum = 5, count = 2

Iteration Count: 0

- 0: sum = 1, count = 0
- 1: sum = 3, count = 1
- 2: sum = 5, count = 2

Iteration Count: 0

- 0: sum = 1, count = 0
- 1: sum = 3, count = 1
- 2: sum = 5, count = 2

Iteration Count: 0

- 0: sum = 1, count = 0
- 1: sum = 3, count = 1
- 2: sum = 5, count = 2
Tracing Loops

```c
int sum = 1;
int count = 0;
while (count < 3) {
    sum = sum + 2;
    count = count + 1;
}
```

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Count &lt; 6</th>
<th>count</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>
Exercise: Tracing loops

```c
int sum = 10;
int count = 0;
while (count < 6) {
    sum = sum - 1;
    count = count + 2;
}
```
Accumulator pattern

Idea: Repeatedly update a variable (typically in a loop)

Pattern:
1. Initialize accumulator variable
2. Loop until done
   1. Update the accumulator variable
Convenience syntax: Assignment

Because updating variable values is so common, language such as Java provide shorthand syntax for it
  • Analogy: contractions in English

    sum = sum + 2
    count = count + 1
    count = count – 1
    product = product * 2
    divisor = divisor / 2
    message = message + “lol!”
Convenience syntax: Assignment

Because updating variable values is so common, language such as Java provide shorthand syntax for it
  • Analogy: contractions in English

<table>
<thead>
<tr>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum = sum + 2</td>
</tr>
<tr>
<td>count = count + 1</td>
</tr>
<tr>
<td>count = count - 1</td>
</tr>
<tr>
<td>product = product * 2</td>
</tr>
<tr>
<td>divisor = divisor / 2</td>
</tr>
<tr>
<td>message = message + “lol”</td>
</tr>
</tbody>
</table>
Convenience syntax: Assignment

Because updating variable values is so common, language such as Java provide shorthand syntax for it

- Analogy: contractions in English

<table>
<thead>
<tr>
<th>Original Syntax</th>
<th>Shorthand Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum = sum + 2</td>
<td>sum += 2</td>
</tr>
<tr>
<td>count = count + 1</td>
<td>count += 1</td>
</tr>
<tr>
<td>count = count - 1</td>
<td>count -= 1</td>
</tr>
<tr>
<td>product = product * 2</td>
<td>product *= 2</td>
</tr>
<tr>
<td>divisor = divisor / 2</td>
<td>divisor /= 2</td>
</tr>
<tr>
<td>message = message + “ lol”</td>
<td>message += “ lol”</td>
</tr>
</tbody>
</table>
Exercise: Write a program that computes powers of 2

Write a program, LoopPow2.java, that computes powers of twos. For example,

```
$ java LoopPow2
Enter an exponent: 0
2 to the power of 0 is 1

$ java LoopPow
Enter an exponent: 1
2 to the power of 1 is 2

$ java LoopPow
Enter an exponent: 4
2 to the power of 4 is 16
```
Example: For Loop

```java
int val = 0;
String valStr = "";
int sum = 0;

for (int count = 0; count < 6; count = count + 1) {
    System.out.print("Enter a number: ");
    valStr = System.console().readLine();
    val = Integer.parseInt(valStr);
    sum = sum + val;
}
System.out.println("The sum is "+sum);
```
Exercise: Tracing loops

```java
String pattern = "";
for (int i = 0; i < 3; i++) {
    pattern = pattern + "*";
}
System.out.println(pattern);
```
Exercise: LoopPattern

```
$ java LoopPattern
Enter a length: 5
*_*_*

$ java LoopPattern
Enter a length: 10
*_*_*_*_*_*

$ java LoopPattern
Enter a length: 0

$ java LoopPattern
Enter a length: 1
* 
```
Exercise: Nested loops

$ java Square
Enter a size: 5
*****
*****
*****
*****
*****

$ java Square
Enter a size: 1
*

$ java Square
Enter a size: 0