CS 113 – Computer Science I

Lecture 06 – Booleans & Conditionals

Thursday 09/21/2023
Announcements

• HW01:
  • Grading should be done by tomorrow

• HW02 – released
  • Due Tuesday 09/26

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

Review
Con
Unit testing

Verify that method is implemented correctly

Call the method with different inputs and check the results

In a library, we can use the main method to test methods
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
Booleans & Conditionals
A new data type: Booleans

• Contains two possible values:
  • true; false;

• bool isWet = true;

• Conditional expression
Conditional Expressions & Relational Operators

• Conditional expression produces either true or false

• Relational Operators:
  • >
  • >=
  • <
  • <=
  • ==
  • !=

• Watch out about == vs =
Exercise: relational expressions

int temp = 68;
double val = 10.5;
boolean raining = true;

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>temp &gt; 80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val != 5.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>val &gt;= 10.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>raining == true</td>
<td></td>
<td></td>
</tr>
<tr>
<td>raining</td>
<td></td>
<td></td>
</tr>
<tr>
<td>raining == false</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Logical Operators

• Way to combine Boolean expressions

• logical Operators:
  • `&&` - and
  • `||` - or
  • `!` - not
Rules of logical operators

1. $X \text{ && } Y$ is true when
   1. Both $X$ and $Y$ are true

2. $X \text{ || } Y$ is true when
   1. $X$ is true or $Y$ is true

3. $!X$ is true when
   1. $X$ is false

4. $!X$ false when
   1. $X$ is true
Exercise: logical expressions

boolean isHappy = true;
boolean knowIt = false;
int temp = 40;

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>isHappy &amp;&amp; knowIt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isHappy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isHappy</td>
<td></td>
<td>temp &gt; 80</td>
</tr>
<tr>
<td>isHappy</td>
<td></td>
<td>knowIt</td>
</tr>
<tr>
<td>!knowIt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isHappy &amp;&amp; (temp &lt; 80</td>
<td></td>
<td>!knowIt)</td>
</tr>
</tbody>
</table>
Decision making: if/else

Idea: Branching decision-making based on Boolean expressions
• Example: A decision tree for Happy.java

```java
if (isHappy && knowIt) {
    System.out.println("Clap your hands!");
} else {
    System.out.println("Sit quietly.");
}
```

![Decision Tree Diagram]

- **isHappy && knowIt**
- **True** -> “Clap your hands”
- **False** -> “Sit Quietly”
Exercise: IsEven

Write a program IsEven which asks the user for an integer and prints whether it is even or not

```
$ java IsEven
Enter an integer: 4
4 is even!

$ java IsEven
Enter an integer: -1
-1 is odd!

$ java IsEven
Enter an integer: 0
0 is even!
```
Decision making: multi-way if statements

if (<condition1>) {
    <stmts>
} else if (<condition2>) {
    <stmts>
}
....
else {
    <stmts>
}

NOTES:
• Conditions evaluated in order
• First true condition executes
• Only one of the conditions can execute!
• the final else statement is optional
Example: Height.java

• Write a program (called Height.java) that determines if a user can ride a rollercoaster.

• Make sure to ask the user for height in inches.
• Prints out a message if they are taller than 5, 4, 3 feet or are too short for the ride
class CheckHeight2 {
    public static void main(String[] args) {
        System.out.print("Enter a height (inches): ");
        int h = Integer.parseInt(System.console().readLine());

        if (h > 36) {
            System.out.println("Taller than 3 ft");
        } else if (h > 60) {
            System.out.println("Taller than 5 ft");
        } else if (h > 48) {
            System.out.println("Taller than 4 ft");
        } else {
            System.out.println("Too small for this ride");
        }
    }
}
class CheckHeight2 {
    public static void main(String[] args) {
        System.out.print("Enter a height (inches): ");
        int h = Integer.parseInt(System.console().readLine());

        if (h > 36) {
            System.out.println("Taller than 3 ft");
        } else if (h > 60) {
            System.out.println("Taller than 5 ft");
        } else if (h > 48) {
            System.out.println("Taller than 4 ft");
        } else {
            System.out.println("Too small for this ride");
        }
    }
}
class CheckHeight2 {
    public static void main(String[] args) {
        System.out.print("Enter a height (inches): ");
        int h = Integer.parseInt(System.console().readLine());
        if (h > 36) {
            println("Taller than 3 ft");
        } else if (h > 60) {
            println("Taller than 5 ft");
        } else if (h > 48) {
            println("Taller than 4 ft");
        } else {
            println("Too small for this ride");
        }
    }
}

Draw the decision tree for this if statement

What is the output of this program:
- if the user enters 62 inches?
- if the user enters 10 inches?
Exercise: Blackjack

Write a program Blackjack.java which generates a random value between 2 and 21

• If the value is 21, print the value and “Blackjack” to the console
• If the value is between 17 and 20, print the value and “Stand” to the console
• If the value is less than 17, print the value and “Hit me!” to the console
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.
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**

```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
| Dec | Hx | Oct | Char   | Dec | Hx | Oct | Html | Char   | Dec | Hx | Oct | Html | Char   | Dec | Hx | Oct | Html | Char   |
|-----|----|-----|--------|-----|----|-----|------|--------|-----|----|-----|------|--------|-----|----|-----|------|--------|-----|----|-----|------|--------|
| 0   | 0  | 000 | NUL    | 32  | 20 | 040 | 262; Space | 64  | 40 | 100 | 164 | 0    | 96  | 60 | 140 | 196 | 0    |
| 1   | 1  | 001 | SOH    | 33  | 21 | 041 | 263; !   | 65  | 41 | 101 | 165 | A    | 97  | 61 | 141 | 197 | a    |
| 2   | 2  | 002 | STX    | 34  | 22 | 042 | 264; "   | 66  | 42 | 102 | 166 | B    | 98  | 62 | 142 | 198 | b    |
| 3   | 3  | 003 | ETX    | 35  | 23 | 043 | 265; #   | 67  | 43 | 103 | 167 | C    | 99  | 63 | 143 | 199 | c    |
| 4   | 4  | 004 | EOT    | 36  | 24 | 044 | 266; $   | 68  | 44 | 104 | 168 | D    | 100 | 64 | 144 | 200 | d    |
| 5   | 5  | 005 | ENQ    | 37  | 25 | 045 | 267; %   | 69  | 45 | 105 | 169 | E    | 101 | 65 | 145 | 201 | e    |
| 6   | 6  | 006 | ACK    | 38  | 26 | 046 | 268; &   | 70  | 46 | 106 | 170 | F    | 102 | 66 | 146 | 202 | f    |
| 7   | 7  | 007 | BEL    | 39  | 27 | 047 | 269; '   | 71  | 47 | 107 | 171 | G    | 103 | 67 | 147 | 203 | g    |
| 8   | 8  | 010 | BS     | 40  | 28 | 050 | 270; (   | 72  | 48 | 110 | 172 | H    | 104 | 68 | 150 | 204 | h    |
| 9   | 9  | 011 | TAB    | 41  | 29 | 051 | 271; )   | 73  | 49 | 111 | 173 | I    | 105 | 69 | 151 | 205 | i    |
| 10  | A  | 012 | LF     | 42  | 30 | 052 | 272; *   | 74  | 50 | 112 | 174 | J    | 106 | 70 | 152 | 206 | j    |
| 11  | B  | 013 | VT     | 43  | 31 | 053 | 273; +   | 75  | 51 | 113 | 175 | K    | 107 | 71 | 153 | 207 | k    |
| 12  | C  | 014 | FF     | 44  | 32 | 054 | 274; -   | 76  | 52 | 114 | 176 | L    | 108 | 72 | 154 | 208 | l    |
| 13  | D  | 015 | CR     | 45  | 33 | 055 | 275; .   | 77  | 53 | 115 | 177 | M    | 109 | 73 | 155 | 209 | m    |
| 14  | E  | 016 | SO     | 46  | 34 | 056 | 276; /   | 78  | 54 | 116 | 178 | N    | 110 | 74 | 156 | 210 | n    |
| 15  | F  | 017 | SI     | 47  | 35 | 057 | 277; 0   | 79  | 55 | 117 | 179 | O    | 111 | 75 | 157 | 211 | o    |
| 16  | 10 | 020 | DLE    | 48  | 36 | 060 | 278; 0   | 80  | 56 | 120 | 180 | P    | 112 | 76 | 160 | 212 | p    |
| 17  | 11 | 021 | DC1    | 49  | 37 | 061 | 279; 1   | 81  | 57 | 121 | 181 | Q    | 113 | 77 | 161 | 213 | q    |
| 18  | 12 | 022 | DC2    | 50  | 38 | 062 | 280; 2   | 82  | 58 | 122 | 182 | R    | 114 | 78 | 162 | 214 | r    |
| 19  | 13 | 023 | DC3    | 51  | 39 | 063 | 281; 3   | 83  | 59 | 123 | 183 | S    | 115 | 79 | 163 | 215 | s    |
| 20  | 14 | 024 | DC4    | 52  | 40 | 064 | 282; 4   | 84  | 60 | 124 | 184 | T    | 116 | 80 | 164 | 216 | t    |
| 21  | 15 | 025 | NAK    | 53  | 41 | 065 | 283; 5   | 85  | 61 | 125 | 185 | U    | 117 | 81 | 165 | 217 | u    |
| 22  | 16 | 026 | SYN    | 54  | 42 | 066 | 284; 6   | 86  | 62 | 126 | 186 | V    | 118 | 82 | 166 | 218 | v    |
| 23  | 17 | 027 | ETB    | 55  | 43 | 067 | 285; 7   | 87  | 63 | 127 | 187 | W    | 119 | 83 | 167 | 219 | w    |
| 24  | 18 | 030 | CAN    | 56  | 44 | 070 | 286; 8   | 88  | 64 | 130 | 188 | X    | 120 | 84 | 170 | 220 | x    |
| 25  | 19 | 031 | EM     | 57  | 45 | 071 | 287; 9   | 89  | 65 | 131 | 189 | Y    | 121 | 85 | 171 | 221 | y    |
| 26  | 1A | 032 | SUB    | 58  | 46 | 072 | 288; ;   | 90  | 66 | 132 | 190 | Z    | 122 | 86 | 172 | 222 | z    |
| 27  | 1B | 033 | ESC    | 59  | 47 | 073 | 289; :   | 91  | 67 | 133 | 191 | [   | 123 | 87 | 173 | 223 | {    |
| 28  | 1C | 034 | FS     | 60  | 48 | 074 | 290; <   | 92  | 68 | 134 | 192 | \   | 124 | 88 | 174 | 224 | |    |
| 29  | 1D | 035 | GS     | 61  | 49 | 075 | 291; =   | 93  | 69 | 135 | 193 | |   | 125 | 89 | 175 | 225 | |    |
| 30  | 1E | 036 | RS     | 62  | 50 | 076 | 292; >   | 94  | 70 | 136 | 194 | |   | 126 | 90 | 176 | 226 | |    |
| 31  | 1F | 037 | US     | 63  | 51 | 077 | 293; ?   | 95  | 71 | 137 | 195 | DEL | 127 | 91 | 177 | 227 | DEL |

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

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
```