The ‘while’ Statement

September 27, 2006

Midterm Results

• Average: 86.1
• Median: 97
• Standard Deviation: 24.18
• Maximum: 127 (out of 130)

Top Scores

• Han Lee 127
• Lex Flagel 125
• Mohd Nasrudin 125
• Seung Song 124
• Ogewu Agbese 123
• Kirk Tuxhorn 122

HW4 Questions?

HW 5 is out

• Due next Friday

Quick review of last lecture
Logic of an if statement

- Condition evaluated
- true
- false

Logic of an if-else statement

- Condition evaluated
- true
- false
- statement1
- statement2

The switch Statement

- The general syntax of a switch statement is:
  
  ```java
  switch (expression) {
    case value1 :
      statement-list1
    case value2 :
      statement-list2
    case value3 :
      statement-list3
    case ...  
  }
  ```

- An example of a switch statement:
  
  ```java
  switch (option) {
    case 'A':
      aCount++;  
      break;
    case 'B':
      bCount++;  
      break;
    case 'C':
      cCount++;  
      break;
  }
  ```

Block Statements

- In an if-else statement, the if portion, or the else portion, or both, could be block statements
  
  ```java
  if (total > MAX)  
  {  
    System.out.println("Error!!");
    errorCount++;
  }  
  else  
  {  
    System.out.println("Total: " + total);
    current = total*2;
  }
  ```

- See Guessing.java (page 216)

Other Stuff from Section 5.3
Comparing Data

- When comparing data using boolean expressions, it's important to understand the nuances of certain data types.
- Let's examine some key situations:
  - Comparing floating point values for equality
  - Comparing characters
  - Comparing strings (alphabetical order)
  - Comparing object vs. comparing object references

Comparing Float Values

- You should rarely use the equality operator (==) when comparing two floating point values (float or double).
- Two floating point values are equal only if their underlying binary representations match exactly.
- Computations often result in slight differences that may be irrelevant.
- In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal.

To determine the equality of two floats, you may want to use the following technique:

```java
if (Math.abs(f1 - f2) < TOLERANCE)
    System.out.println("Essentially equal");
```

- If the difference between the two floating point values is less than the tolerance, they are considered to be equal.
- The tolerance could be set to any appropriate level, such as 0.000001.

Comparing Characters

- In Unicode, the digit characters (0-9) are contiguous and in order.
- Likewise, the uppercase letters (A-Z) and lowercase letters (a-z) are contiguous and in order.

<table>
<thead>
<tr>
<th>Characters</th>
<th>Unicode Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 9</td>
<td>48 through 57</td>
</tr>
<tr>
<td>A – Z</td>
<td>65 through 90</td>
</tr>
<tr>
<td>a – z</td>
<td>97 through 122</td>
</tr>
</tbody>
</table>

- As we’ve discussed, Java character data is based on the Unicode character set.
- Unicode establishes a particular numeric value for each character, and therefore an ordering.
- We can use relational operators on character data based on this ordering.
- For example, the character '+' is less than the character 'J' because it comes before it in the Unicode character set.
- Appendix C provides an overview of Unicode.

Comparing Strings

- Remember that in Java a character string is an object.
- The equals method can be called with strings to determine if two strings contain exactly the same characters in the same order.
- The equals method returns a boolean result.

```java
if (name1.equals(name2))
    System.out.println("Same name");
```
Comparing Strings

• We cannot use the relational operators to compare strings.
• The `String` class contains a method called `compareTo` to determine if one string comes before another.
• A call to `name1.compareTo(name2)` returns zero if `name1` and `name2` are equal (contain the same characters).
• It returns a negative value if `name1` is less than `name2`.
• It returns a positive value if `name1` is greater than `name2`.

If (name1.compareTo(name2) < 0)
    System.out.println (name1 + " comes first");
else if (name1.compareTo(name2) == 0)
    System.out.println ("Same name");
else
    System.out.println (name2 + " comes first");

Because comparing characters and strings is based on a character set, it is called a lexicographic ordering.

Lexicographic Ordering

• Lexicographic ordering is not strictly alphabetical when uppercase and lowercase characters are mixed.
• For example, the string “Great” comes before the string “fantastic” because all of the uppercase letters come before all of the lowercase letters in Unicode.
• Also, short strings come before longer strings with the same prefix (lexicographically).
• Therefore “book” comes before “bookcase”.

Comparing Objects

• The `==` operator can be applied to objects – it returns true if the two references are aliases of each other.
• The `equals` method is defined for all objects, but unless we redefine it when we write a class, it has the same semantics as the `==` operator.
• It has been redefined in the `String` class to compare the characters in the two strings.
• When you write a class, you can redefine the `equals` method to return true under whatever conditions are appropriate.

Repetition Statements

• Repetition statements allow us to execute a statement multiple times.
• Often they are referred to as loops.
• Like conditional statements, they are controlled by boolean expressions.
• Java has three kinds of repetition statements:
  • the `while` loop
  • the `do` loop
  • the `for` loop
• The programmer should choose the right kind of loop for the situation.
The while Statement

- A while statement has the following syntax:

```java
while (condition) 
{ 
statement; 
}
```

- If the condition is true, the statement is executed.
- Then the condition is evaluated again, and if it is still true, the statement is executed again.
- The statement is executed repeatedly until the condition becomes false.

An example of a while statement:

```java
int count = 1;
while (count <= 5) 
{ 
System.out.println (count);
  count++; 
}
```

- If the condition of a while loop is false initially, the statement is never executed.
- Therefore, the body of a while loop will execute zero or more times.

Let's look at some examples of loop processing:

- A loop can be used to maintain a running sum.
- A sentinel value is a special input value that represents the end of input.
- See Average.java (page 229)

- A loop can also be used for input validation, making a program more robust.
- See WinPercentage.java (page 231)

Example: Average.java (page 229)
Example: **WinPercentage.java** (page 231)

**Infinite Loops**

- The body of a **while** loop eventually must make the condition false.
- If not, it is called an **infinite loop**, which will execute until the user interrupts the program.
- This is a common logical error.
- You should always double check the logic of a program to ensure that your loops will terminate normally.

An example of an infinite loop:

```java
int count = 1;
while (count <= 25) {
    System.out.println (count);
    count = count - 1;
}
```

This loop will continue executing until interrupted (Control-C) or until an underflow error occurs.

**Nested Loops**

- Similar to nested **if** statements, loops can be nested as well.
- That is, the body of a loop can contain another loop.
- For each iteration of the outer loop, the inner loop iterates completely.
- See **PalindromeTester.java** (page 235)

**Nested Loops**

- How many times will the string "Here" be printed?

```java
count1 = 1;
while (count1 <= 10) {
    count2 = 1;
    while (count2 <= 20) {
        System.out.println ("Here");
        count2++;
    }
    count1++;
}
```

10 * 20 = 200

THE END