CS1004: Intro to CS in Java, Spring 2005

Lecture #16: Java conditionals/loops, cont'd.

Janak J Parekh
janak@cs.columbia.edu

Administrivia

- Midterms returned now
- Weird distribution
- Mean: 35.4 ± 8.4
- What does this mean?
- In-class extra credit
- HW#3 returned Thurs.
- HW#4 out

The Conditional Operator, redux

- (Another) Example:

  System.out.println("Your change is "+count+
  ((count == 1) ? "Dime" : "Dimes");

- If `count` equals 1, then "Dime" is printed
- If `count` is anything other than 1, then "Dimes" is printed
The switch Statement

- The switch statement provides another way to decide which statement to execute next.
- The switch statement evaluates an expression, then attempts to match the result to one of several possible cases.
- Each case contains a value and a list of statements.
- The flow of control transfers to statement associated with the first case value that matches.

The switch Statement (II)

- The general syntax of a switch statement is:

```java
switch (expression)
{
    case value1:
        statement-list1
    case value2:
        statement-list2
    case value3:
        statement-list3
    case ...
}
```

switch and break

- Often a break statement is used as the last statement in each case's statement list.
- A break statement causes control to transfer to the end of the switch statement.
- If a break statement is not used, the flow of control will continue into the next case.
- Biggest common bug with switch, and a reason why I use it sparingly.
switch Example

- An example of a switch statement:

```
switch (option) {
    case 'A':
        aCount++;
        break;
    case 'B':
        bCount++;
        break;
    case 'C':
        cCount++;
        break;
}
```

switch and default case

- A switch statement can have an optional default case
- The default case has no associated value and simply uses the reserved word `default`
- If the default case is present, control will transfer to it if no other case value matches
- If there is no default case, and no other value matches, control falls through to the statement after the switch

What can you switch on?

- The expression of a switch statement must result in an integral type, meaning an integer (byte, short, int, long) or a char
- It cannot be a boolean value or a floating point value (float or double)
- The implicit boolean condition in a switch statement is equality (==, not .equals())
- Common for things like menu systems (“Enter one of the above 5 options”)
Comparing Data

- When comparing data using boolean expressions, it’s important to understand the nuances of certain data types
- We’ve talked about these, but now let’s formalize it

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

Comparing Float Values (II)

- 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

- 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 Characters (II)

- 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>

String equality

- 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");
```
**String inequalities**

- 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`.

**compareTo example**

```java
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, as we mentioned before.
- The `equals` method is also 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 referred to as loops.
- Like conditional statements, they are controlled by boolean expressions.
- Java has three kinds of repetition statements: `while`, `do`, and `for`.
  - All are equivalent, but some are easier to use for certain cases.

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.
Logic of a while Loop

Example

- 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.

More complex example

- Maintain a running sum
  - A sentinel value is a special input value that represents the end of input
- Input validation
  - “While the user types an invalid value, reject and wait for a valid value.”
- Example: calculate mean of exams
- Similar to if statements, while statements can be nested as well
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

Infinite Loops

- 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 or until an underflow error occurs

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++;
}
```
break, version 2

- We saw `break` in the context of `switch`, but it can be used with `while` (and other loops) as well; for example,

  ```java
  while(true) {
    if(i > 10) break;
    else i++;
  }
  ```

- What is this code equivalent to?
- Generally, you don’t use `break`, but it’s useful to have, especially if the while loop is very complex.
- If you have nested loops, `break` only breaks out of the most immediate loop, not all of them.
- `return` can be used to break out of a bunch of loops, but avoid

The do Statement

- A `do` statement has the following syntax:

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

- The `statement` is executed once initially, and then the `condition` is evaluated.
- The statement is executed repeatedly until the condition becomes false.

The do Statement

- An example of a `do` loop:

  ```java
  int count = 0;
  do {
    count++;
    System.out.println(count);
  } while (count < 5);
  ```

- The body of a `do` loop executes at least once.
- What’s the result of this code fragment?
- `do` is particularly useful for “interactive repetition”
Comparing while and do

The while Loop

condition evaluated
true
false
statement

The do Loop

condition evaluated
true
false
statement

The for Statement

- A for statement has the following syntax:

```
for ( initialization ; condition ; increment )
statement;
```

- The **initialization** is executed once before the loop begins.
- The **statement** is executed until the **condition** becomes false.
- The **increment** portion is executed at the end of each iteration.

Logic of a for loop

```
initialization

condition evaluated
true
false
statement
increment
```
The for Statement

- A for loop is functionally equivalent to the following while loop structure:

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

An example of a for loop:

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

The initialization section can be used to declare a variable.
Like a while loop, the condition of a for loop is tested prior to executing the loop body.
Therefore, the body of a for loop will execute zero or more times.

The for Statement

- The increment section can perform any calculation.

```java
for (int num=100; num > 0; num -= 5)
    System.out.println(num);
```

- A for loop is well suited for executing statements a specific number of times that can be calculated or determined in advance.
The for Statement

- Each expression in the header of a for loop is optional
- If the initialization is left out, no initialization is performed
- If the condition is left out, it is always considered to be true, and therefore creates an infinite loop
- If the increment is left out, no increment operation is performed

Next time

- Finish for
- Start building more complex examples with loop constructs
- Think about how algorithms are created using conditions and loops