Administrivia

- Homework due now
- Midterm on Thursday
  - We’ll stop the lecture at about noon and I’ll take questions at that point

Conditional Statements

- A conditional statement lets us choose which statement will be executed next
- The Java conditional statements are the:
  - if statement
  - if-else statement
  - ? operator (well, not quite a statement)
  - switch statement
- Less “clumsy” than the assembly equivalents
The if Statement

- The if statement has the following syntax:

```
if (condition) 
statement;
```

- The condition must be a boolean expression. It must evaluate to either true or false.
- If the condition is true, the statement is executed.
- If it is false, the statement is skipped.

Boolean Expressions

- Java's equality operators or relational operators all return boolean results:
  - `==` equal to
  - `!=` not equal to
  - `<` less than
  - `>` greater than
  - `<=` less than or equal to
  - `>=` greater than or equal to
- Remember, equality operator (`==`) vs. assignment operator (`=`)
- Lower precedence than math operators

The if Statement

- An example of an if statement:
  ```java
  if (sum > MAX) 
  delta = sum - MAX; 
  System.out.println("The sum is " + sum);
  ```
- First the condition is evaluated -- the value of `sum` is either greater than the value of MAX, or it is not
- If the condition is true, the assignment statement is executed -- if it isn’t, it is skipped.
- Either way, the call to `println` is executed next
Indentation

- The statement controlled by the if statement is indented to indicate that relationship.
- The use of a consistent indentation style makes a program easier to read and understand.
- Although it makes no difference to the compiler, proper indentation is crucial when the code needs to be maintained.
- Emacs will do this automatically for you; just hit TAB once.

Logical Operators

- Boolean expressions can also use the following logical operators:
  
  - `!` Logical NOT
  - `&&` Logical AND
  - `||` Logical OR

- Exactly like circuit/assembly equivalents.
- Process boolean operands, and produce boolean results.

Logical Operators (II)

- Expressions that use logical operators can form complex conditions.

  ```java
  if (total < MAX+5 && !found)
  System.out.println("Processing...");
  ```

- All logical operators have lower precedence than the relational operators (and math operators).
- Personally, I would use parentheses.
- Logical NOT has higher precedence than logical AND and logical OR.
Short-Circuited Operators

- The processing of logical AND and logical OR is “short-circuited”
- If the left operand is sufficient to determine the result, the right operand is not evaluated

```java
if (count != 0 && total/count > MAX)
    System.out.println("Testing...");
```

- This type of processing must be used carefully

if-else

- An else clause can be added to an if statement to make an if-else statement

```java
if (condition)
    statement1;
else
    statement2;
```

- If the condition is true, statement1 is executed; if the condition is false, statement2 is executed
- One or the other will be executed, but not both

Indentation Revisited

- Remember that indentation is for the human reader, and is ignored by the computer
- Emacs will help you avoid this confusion

```java
if (total > MAX)
    System.out.println("Error!!");
    errorCount++;
```

Despite what is implied by the indentation, the increment will occur whether the condition is true or not
Block Statements

- Several statements can be grouped together into a block statement delimited by braces
- A block statement can be used wherever a statement is called for in the Java syntax rules
- Bracing can be spaced in different ways (book uses open bracing, I use closed bracing)

```java
if (total > MAX)
{
    System.out.println ("Error!!");
    errorCount++;
}
```

Block Statements (II)

- 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;
}
```

When in doubt, brace!

- It’s okay to use braces even when you have one statement
- I’ll almost always use braces, and will only occasionally omit them
Else if

- We can have more than two conditions:
  ```java
  if(age < 20) {
    System.out.println("You're young!");
  } else if(age > 20 && age < 40) {
    System.out.println("You're not so young!");
  } else if(age > 40 && age < 60) {
    System.out.println("You're a bit older!");
  } else {
    System.out.println("You're still a student?");
  }
```
- Starts with the top clause and works down from there
- Last else is only run if none of the others matched
- Major bug(s) in this code; what is it?

Nested if Statements

- The statement executed as a result of an if statement or else clause could be another if statement
- These are called nested if statements
- An else clause is matched to the last unmatched if (no matter what the indentation implies)
- Braces can be used to specify the if statement to which an else clause belongs
- Not the same thing as else if

Let's put it all together...

- Modify our DieRoller class to ask the user to guess the value of the die
The Conditional Operator

- Java has a *conditional operator* that uses a boolean condition to determine which of two expressions is evaluated.
- Its syntax is:  
  \[ \text{condition} \ ? \ \text{expression1} \ : \ \text{expression2} \]
- If the *condition* is true, *expression1* is evaluated; if it is false, *expression2* is evaluated.
- The value of the entire conditional operator is the value of the selected expression.
- Sometimes called an “immediate if”

The Conditional Operator (II)

- The conditional operator is similar to an `if-else` statement, except that it is an expression that returns a value.
- For example:
  \[ \text{larger} = ((\text{num1} > \text{num2}) \ ? \ \text{num1} \ : \ \text{num2}); \]
- If `num1` is greater than `num2`, then `num1` is assigned to `larger`; otherwise, `num2` is assigned to `larger`.
- The conditional operator is *ternary* because it requires three operands.
- Use parentheses to avoid confusion.

The Conditional Operator (III)

- Another example:
  ```java
  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 `case` are reserved words
- If `expression` matches `value2`, control jumps to here
- 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 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:

```java
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:
  
  ```
  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)
  - returns a negative value if `name1` is less than `name2`
  - returns a positive value if `name1` is greater than `name2`

```java
compareTo example

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.

Midterm exam

- Three parts
  - True/False (4-5 questions)
  - Short answer (3-4 questions)
  - Long answer (one question)
- Covers lectures 1-14, S/G ch. 1, 4 and 5, and L/L ch. 1-4
- Except stuff at the very end of lecture 14 (if statements)

Sample T/F question

- In this section, assert whether the proposition is true or false, and provide a one-sentence justification as to why. (If you feel an assertion is ambiguous, review the course materials: it will have been well-defined somewhere.)
- You run a Java program on CUNIX by typing java Foo.java at the $ prompt and hitting Enter.
Sample short-answer question

- State two advantages and two disadvantages of using applets as opposed to applications.
- Some short-answer questions may be more structured than others:
  - You’re given the following piece of code. Explain what it does.

Sample long-answer question

- (L/L exercise 4.1) Write a method called `randomInRange` that accepts two integer parameters representing a range. The method should return a random integer in the specified range (inclusive). Assume that the first parameter is greater than the second.
- Well, this one is only 1 line of code, so it might be a bit longer
- By the way, I will ask theory questions: these are just examples

Next time

- Exam. ☄
- After break, finish chapter 5 of L/L
  - Loops