Boolean Expressions and If

- Flow of Control / Conditional Statements
- The if Statement
- Logical Operators
- The else Clause
- Block statements
- Nested if statements
- Reading for this class: L&L, 5.1 – 5.2

Flow of Control

- Default order of statement execution is linear: one after another in sequence
- But, sometimes we need to decide **which** statements to execute and/or **how many times**
- These decisions are based on *boolean expressions* (or “conditions”) that evaluate to **true** or **false**
- The resulting order of statement execution, according to these decisions, is called the *flow of control*

Conditions/Boolean Expressions

- A condition is often expressed as a **boolean expression** (which returns a boolean result).
- Boolean expressions, like arithmetic ones, use operators, such as the following **equality** and **relational** operators:

==	equal to
!=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

- **Note**: $\boxed{==}$ and $\boxed{=}$ are **not** the same!

Boolean Expressions

- 5 < 7
- 7 >= 5
- --x == 98
- offer < minimumBid
- grade+1 >= aGrade
- tWeight < weight++
- password.length() >= MIN_LENGTH
- insPremium * months != benefits – deductible
- (volume – (1 / pHValue)) * 2 <= 1 / qFactor
- a-- * (b / ((c – d) % e)) == (b * (c / a) + ((3 % q) + 7)

Logical Operators

- The following *logical operators* can also be used in boolean expressions:

`!`	Logical NOT
`&&`	Logical AND
`||`	Logical OR

- They operate on <u>boolean operands</u> and produce <u>boolean results</u>
  - Logical NOT is a **<u>unary</u>** operator => **<u>one operand</u>**
  - AND and OR are **<u>binary</u>** operators => **<u>two operands</u>**

Logical NOT

- The *logical NOT* operation is also called *logical negation* or *logical complement*
- If some boolean condition `a` is true, then `!a` is false;
- If `a` is false, then `!a` is true
- Logical operations can be shown with a *truth table*

a	**!a**
true	false
false	true

Logical AND and Logical OR

- The *logical AND* expression

  ```
  a && b
  ```

  is true if **both** `a` and `b` are true, and false otherwise

- The *logical OR* expression

  ```
  a || b
  ```

  is true if **at least** one of `a` or `b` is true, and false otherwise

Logical AND and Logical OR

- The *logical AND* expression

```
a && b
```

  is true if **both** `a` and `b` are true, and false otherwise

- The *logical OR* expression

```
a || b
```

  is true if **at least** one of `a` or `b` is true, and false otherwise

Logical Operators

- A truth table shows all possible true–false combinations of the terms
- Since `&&` and `||` each have two operands, there are four possible combinations of conditions `a` and `b`

a	b	a && b	a || b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

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

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

- This coding technique must be used carefully

More Boolean Expressions

```
5 < 7 || offer < minBid          offer >= min || exempt

7 >= 5 && --x == 98              !done && x++ == 47

!(5 < 7 || offer < min) || 7 >= 5 && --x == 98

!(grade+1 >= aGrade) || !(tWeight < weight++)

(!(password.length() >= MIN) || myBoolean) &&
        insPremium * months != benefits – deductible

(!myBoolean || (volume – (1 / pHValue)) * 2 <= 1 /
qFactor) || !(a-- * (b / ((c – d) % e)) == (b * (c / a)
+ ((3 % q) + 7))
```

Conditional Statements

- A *conditional statement* decides which program statement will be executed next
- We use conditional statements to make basic decisions as the program runs.
- Recall the Quadratic example:
  - Check if a = 0, if b = 0, etc.
- The Java conditional statements are the:
  - *if statement*
  - *if-else statement*
  - *switch statement*

The if Statement

- The *if statement* has the following syntax:

`if` is a Java reserved word

The *`condition`* must be a boolean expression. It must **evaluate** to either true or false.

```
if ( condition ){
    statement;
}
```

If the *`condition`* is true (i.e., **evaluates** to true), the *`statements`* are executed.
If it is false, the *`statements`* are skipped.

The if Statement

- An example of an `if` statement:

```
if (sum > MAX){
   delta = sum - MAX;
}
System.out.println ("The sum is " + sum);
```

- First the condition is evaluated -- either 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 false, it is not
- The `println`, **not** being contingent upon `sum < MAX`, is always executed next

Indentation

- The statement controlled by the `if` statement is **indented** to indicate that relationship

```
if (sum > MAX){
   delta = sum - MAX;
}
System.out.println ("The sum is " + sum);
```

- A consistent indentation style makes a program easier to read and understand
- The compiler doesn't care about proper indentation, **but human readers do!**

Block Statements

- Several statements can be placed between the braces – called a "block statement"

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

- A block statement can be used to treat **several** statements as **one**
- "if true..."
  - one statement => "do ***this thing***"
  - 2 or more => "do ***this group*** of things"

The if-else Statement

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

```
if ( condition ){
    statement1;
}
else{
    statement2;
}
```

`condition` is true => `statement1` is executed

`condition` is false => `statement2` is executed

- One or the other will be executed, but not both
- See Wages.java (page 217)

Block Statements

- In an `if-else` statement, the `if` portion, or the `else` portion, or both, could be block statements

```
if (total > MAX)
{
   System.out.println ("Error!!");
   errorCount++;
}
else
{
   System.out.println ("Total: " + total);
   current = total*2;
}
```

Composing an if(-else) statement

```
if (offer < minimumBid){
   System.out.println("Offer is too low. " +
       "Please bid at least $" + minimumBid);
   offer = scan.nextDouble();
   System.out.println("You bidded $" + offer);
}
else {
   System.out.println("You bidded $" + offer);
   System.out.println("Raise your offer to the " +
     "current highest? YES or NO");
   answer = scan.nextLine();
}
```

Omitting Braces

- If you have only one statement after the if or the else, braces are not strictly necessary:

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

- However, you must know what you're doing.

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

Despite the indentation, the increment will occur regardless:

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

- Thus, consider using braces every time until you know what you're doing.

Nested if Statements

- An `if` statement or an `else` clause can contain ***another*** conditional statement
- The inner if statement is treated as a single statement, but...
- An `else` clause is matched to the last unmatched `if` by default, unless...
- **Braces** are used to specify the `if` statement to which an `else` clause belongs

```
int num = 3;
if (num > 2) {
  if (num > 4)
    System.out.println("num > 2 " +
        "num > 4");

}
else
  System.out.println("num <= 2");
-> num <= 2
```

Demo: MinOfThreeAlt.java

In addition, you also have the if/else-if/else format. Demo: If_ElseIf_ElseDemo.java

The Conditional Operator

- Java has a *conditional operator* that uses a boolean condition to evaluate one of two expression
- Its syntax is:

```
condition ? expression1 : 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**

The Conditional Operator

- The conditional operator is similar to an `if-else` statement, except that it is an expression that returns a single value

- For example, these are functionally equivalent:

```
larger = ((num1 > num2) ? num1 : num2);

  if (num1 > num2)
    larger = num1;
  else
    larger = num2;
```

- The conditional operator is *ternary* because it requires three operands: a condition and two alternative values

Project 1 Application

- Now, you have been shown how to use boolean operators and `if` and `if-else` statements
- You need to use the appropriate nested if statements and else clauses in the `doRoll()` method
- Specifically, you will need to examine the scoring rules for the dice game and translate them into a set of (nested) `if-else` statements