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

- Unless specified otherwise, the order of statement execution through a method is linear:
  - one statement after another in sequence
- Some programming statements allow us to:
  - decide whether or not to execute a particular statement
  - execute a statement over and over, repetitively
- These decisions are based on *boolean expressions* (or *conditions*) that evaluate to true or false
- The order of statement execution is called the *flow of control*

Conditions/Boolean Expressions

- A condition is often obtained using an *equality operator and/*or *relational operator* which create boolean expressions that return boolean results:

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

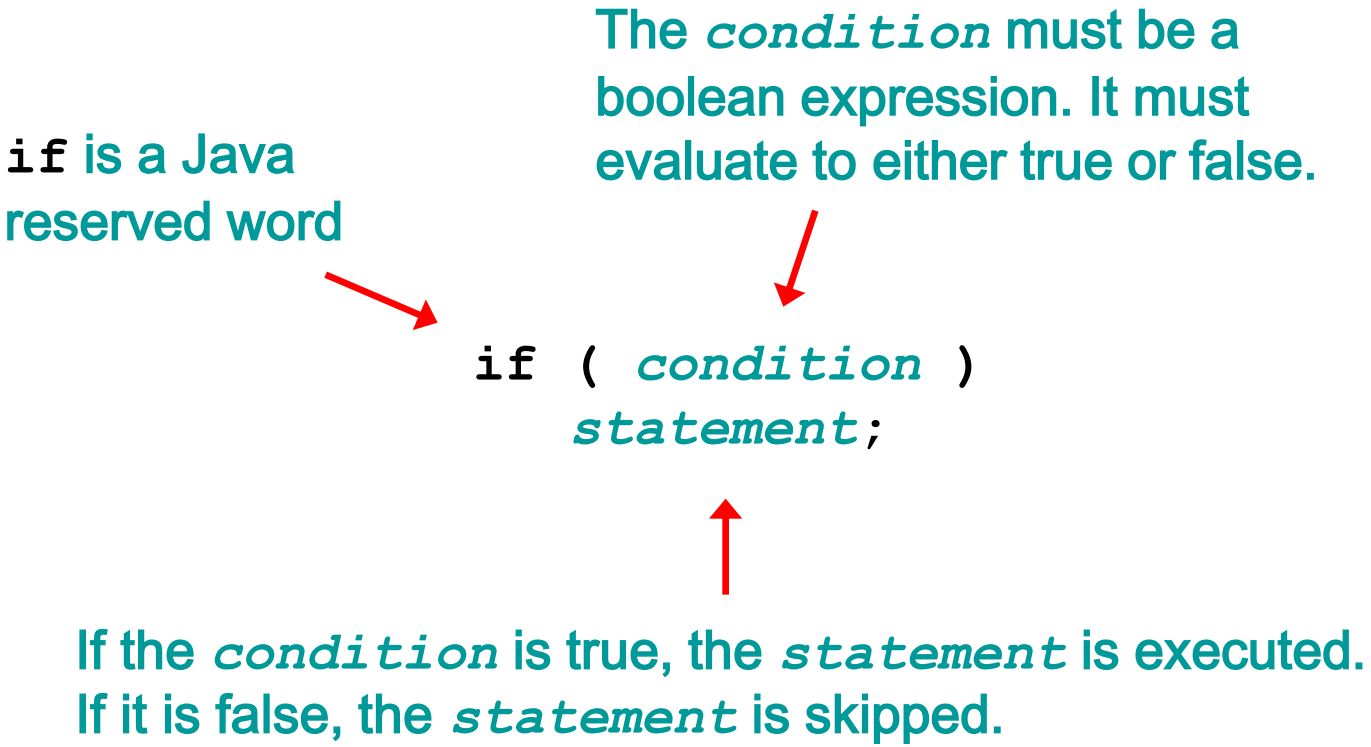
- Note the difference between the equality operator (==) and the assignment operator (=)

Conditional Statements

- A *conditional statement* lets us choose which statement will be executed next
- Therefore they are sometimes called *selection statements*
- Conditional statements give us the power to make basic decisions
- The Java conditional statements are the:
  - *if statement*
  - *if-else statement*
  - *switch statement*

The if Statement

- The *if statement* has the following syntax:



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 -- 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 true, it is skipped.
- Either way, the call to `println` is executed next
- See Age.java (page 214-215)

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 to human readers

Logical Operators

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

!	Logical NOT
&&	Logical AND
||	Logical OR

- They operate on boolean operands and produce boolean results
  - Logical NOT is a unary operator (it operates on one operand)
  - Logical AND and logical OR are binary operators (each operates on two operands)

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 `a` or `b` or both are 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

The if-else Statement

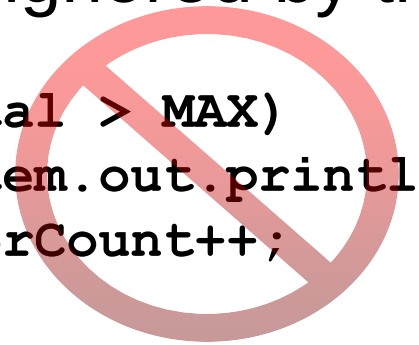
- An *else clause* can be added to an `if` statement to make an *if-else statement*

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

- 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
- See Wages.java (page 217)

Indentation Revisited

- Remember that indentation is for the human reader and is ignored by the Java compiler



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

Despite what is implied by the indentation, the increment will occur whether the if condition is true or not, as follows:

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

Block Statements

- Several statements can be grouped into a *block statement* delimited by braces

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

Now the increment will only occur when the if condition is true

- A block statement can be used wherever a statement is called for in the Java syntax

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;
}
```

The Conditional Operator

- Java has a *conditional operator* that uses a boolean condition to determine which of two expressions is evaluated
- 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:

```
larger = ((num1 > num2) ? num1 : 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: a condition and two alternative values

Nested if Statements

- The statement executed as a result of an `if` statement or an `else` clause can 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
- See MinOfThree.java (page 225)

Nested Conditional Operators

- Alternative MinOfThree.java

```
Scanner scan = new Scanner (System.in);

System.out.println ("Enter three integers: ");
int num1 = scan.nextInt();
int num2 = scan.nextInt();
int num3 = scan.nextInt();

int min = (num1 < num2) ?
            ((num1 < num3) ?  num1 : num3) :
            ((num2 < num3) ?  num2 : num3);

System.out.println ("Minimum value: " + min);
```

Project 1 Application

- Now, you have been shown the Java statements that you will need to use for checking the values of “a”, “b”, “c”
- You need to use the appropriate nested if statements and else clauses in your getSolution () method

Project 1 Application

- Conditions that may be useful in Project 1

```
a == 0  // true when a is equal to zero
```

or

```
a == 0 && b == 0 && c == 0  // true when
  all of them are zero
```

- Put one of those boolean expressions inside the parentheses within an if statement

```
if (a == 0)
```

or

```
if (a == 0 && b == 0 && c == 0)
```

Project 1 Application

- Conditions that may be useful in Project 1

```
a <= 0  // true when a is negative/zero
```

or

```
a <= 0 || b <= 0 || c <= 0  // true when
  any of them are negative/zero
```

- Put one of those boolean expressions inside the parentheses within an if statement

```
if (a <= 0)
```

or

```
if (a <= 0 || b <= 0 || c <= 0)
```