Computer Science I

Chapter 3 Notes

Conditionals and Loops

Now we will examine programming statements that allow us to:

- make decisions
- work with Strings

Key concepts

- boolean expressions
- conditional statements
- comparing primitive data values
- String methods
- comparing Strings

Flow of Control

- Unless specified otherwise, program statements are executed one statement after another, in sequence.
- Conditional statements allow us to decide whether or not to execute a particular statement.
- 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*.

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:
  - *if statement*
  - *if-else statement*
  - *switch statement*

The if Statement

- The <u>if statement</u> has the following syntax:

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

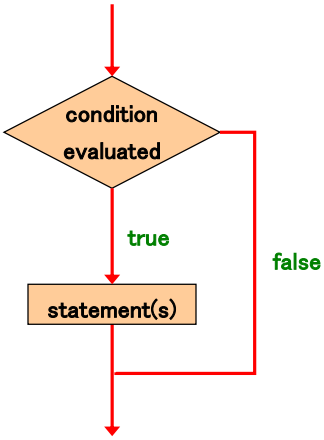
`if` is a Java reserved word

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

There must be parentheses around the condition.

If the *condition* is true, the *statement* is executed.
If it is false, the *statement* is skipped.

Logic of an if Statement



Boolean Expressions

- A condition often uses one of Java's *equality operators* or *relational operators*, which all 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 (==) which requires two equals signs and the assignment operator (=).

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, 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 for readability.

"Always code as if the person who ends up maintaining your code will be a violent psychopath who knows where you live."

-- Martin Golding

THE IF STATEMENT

- What do the following statements do?

```
if (top >= MAXIMUM)
   top = 0;
```

Sets `top` to zero if the current value of `top` is greater than or equal to the value of `MAXIMUM`.

```
if (total != stock + warehouse)
   inventoryError = true;
```

Sets `inventoryError` to true if the value of `total` is not equal to the sum of `stock` and `warehouse`.

The precedence of the arithmetic operators is higher than the precedence of the equality and relational operators

Logical Operators

- Boolean expressions can also use the following *logical operators*:

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

- They all take 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 expressions can be shown using 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

- Expressions that use logical operators can form complex conditions

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

All logical operators have lower precedence than the relational operators.

Logical NOT has higher precedence than logical AND and logical OR.

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

Boolean Expressions

- Specific expressions can be evaluated using truth tables

total < MAX	found	!found	total < MAX && !found
false	false	true	false
false	true	false	false
true	false	true	true
true	true	false	false

THE IF-ELSE STATEMENT

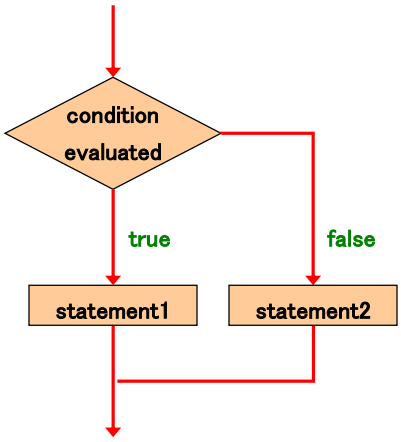
- An *else clause* can be added to an `if` statement to make an *if-else statement*

```
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

LOGIC OF AN IF-ELSE STATEMENT



Indentation Revisited

- Remember that indentation is for the human reader, and is ignored by the computer.

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

Despite what is implied by the indentation, the increment (errorCount++;) 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.

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

- Now that the braces have been added, both the print statement and the increment are executes when the condition is true.

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 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.

The Conditional Operator

- 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.

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.

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 the statement associated with the first case value that matches.

THE SWITCH STATEMENT

- The general syntax of a `switch` statement is:

```
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

THE SWITCH STATEMENT

- 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.
- Sometimes this may be appropriate, but often we want to execute only the statements associated with one case.

The switch Statement

- An example of a switch statement:

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

The switch Statement

- 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.

The switch Statement

- 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.
- You cannot perform relational checks (<, >, <=, or >=) with a `switch` statement.

Comparing Characters

- 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 '+' comes first in the Unicode.

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.

Characters	Unicode Values
0 – 9	48 through 57
A – Z	65 through 90
a – z	97 through 122

- Note that there is a gap between the uppercase and lowercase letters. Codes 91 through 96 are punctuation symbols.

The String Class

- In Java, String is not a primitive type. A character string is an object.
- Because strings are so common, we don't have to use the `new` operator to create a `String` object

```
String title = "Gone With The Wind";
```

- This is special syntax that works <u>only</u> for strings.

3-34

The String Class

- A String literal must be enclosed in double quotes (quotation marks).
- Here are some examples:

```
String word = "banana";
String caption = "A Day at the Park";
```

"banana" and "A Day at the Park" are String literals.

word and caption are references to String objects.

3-35

String Methods

- Once a String object has been created, neither its value nor its length can be changed.
- Thus we say that an object of the String class is immutable.
- However, several methods of the String class return new String objects that are modified versions of the original.
- See sections 3.13 and 3.14 in the textbook.
- For more, see

  https://www.w3schools.com/java/java_ref_string.asp

3-36

String Indexes

- Sometimes we need to refer to a particular character within a String.
- This can be done by specifying the character's numeric *index (or position)*.
- Indexes begin at zero in each String.
- In the String `"Hello"`, the character `'H'` is at index 0 and the `'o'` is at index 4.

3-37

Comparing Strings

- We cannot use the equality operators (==, !=) to compare Strings.
- 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.
- In the following example, name1 and name2 have been declared as type String.

```
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 exactly the same characters)
  - returns a negative value if `name1` is less than `name2`
  - returns a positive value if `name1` is greater than `name2`

Comparing Strings

```
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.
- Therefore `"book"` comes before `"bookcase"`