# Module 2 – Unit 1 Class Notes

# Control Structures I: Selection

## At a Glance

- Overview
- Objectives
- Additional Resources
- Key Terms

# Overview

In this unit, you will learn about control structures `if`, `if…else`, and `switch`. You will learn how to incorporate these into programs to enable them to make decisions, as opposed to simply executing statements in a sequential order.

# Unit Objectives

In this unit, you will:

- Learn about control structures
- Examine relational and logical operators
- Explore how to form and evaluate logical (boolean) expressions
- Learn how to use the selection control structures `if` and `if…else` in a program
- Learn how to avoid bugs by avoiding partially understood concepts and techniques
- Learn how to use the selection control structure `switch` in a program
- Explore how to compare strings

# Unit Tips

## Control Structures

1. A computer can process a program sequentially, by selection, or by repetition.

2. Selection is often called “branching” and repetition is often called “looping.”

## Relational Operators

1. In Java, an expression that has a value of either `true` or `false` is called a logical, or Boolean, expression, and the values `true` and `false` are called logical (Boolean) values.

2. A relational operator is an operator that allows you to make comparisons in a program.

3. Each of the relational operators is a binary operator, and the result of the comparison is either `true` or `false`.

4. In Java, the symbol ==, which consists of two equal signs, is called the equality operator. Recall that the symbol = is called the assignment operator.

5. The equality operator, ==, determines whether two expressions are equal, whereas the assignment operator, =, assigns the value of an expression to a variable.

## Relational Operators and Primitive Data Types

1. Relational operators can be used with integral and floating-point primitive types.

2. For `char` values, whether an expression evaluates to `true` or `false` depends on the collating sequence of the Unicode character set.

3. Comparing the character representation of numbers such as 5 == '5' will evaluate to false because the character '5' represents the Unicode value 53.

## Logical (Boolean) Operators and Logical Expressions

1. Logical operators enable you to combine logical expressions.

2. Logical operators are also referred to as Boolean operators, and logical operators take only logical values as operands and yield only logical values as results.

3. Putting the (`!`) mark in front of a logical expression reverses the value of that logical expression.

## Order of Precedence

1. Relational and logical operators are evaluated from left to right; their associativity is from left to right.

2. Parentheses can be used to change the order in which expressions are evaluated.

### `Boolean` Data Type and Logical (Boolean) Expressions

1. You can manipulate logical expressions using the `boolean` data type and Java reserved words `boolean`, `true`, and `false`.

## Selection: `if` and `if`…`else`

1. In Java, there are three selection or branch control structures: `if` statements, `if`…`else` statements, and the `switch` structures.

### One-Way Selection

1. In Java, one-way selections are incorporated using the if statement.

2. The syntax of one-way selection is:

```
if (logical expression) {
        statement
}
```

### Two-Way Selection

1. To choose between two alternatives, that is, to implement two-way selections, Java provides the if...else statement.

2. Two-way selection uses the following syntax:

```
if (logical expression) {
        statement(s)
}
else {
        statement(s)
}
```

### Compound (Block of) Statements

1. The `if` and `if...else` structures control only one statement at a time, and to permit more complex statements, Java provides a structure called a compound statement or block of statements.

2. Block statements are enclosed in curly braces and consist of a sequence of statements to be executed depending on the evaluation of the `if` and `if…else` expressions.

### Multiple Selections: Nested `if`

1. In a nested `if` statement, Java associates an `else` with the most recent incomplete `if`—that is, the most recent `if` that has not been paired with an `else`.

### Comparing `if...else` Statements with a Series of `if` Statements

1. A series of `if` statements can be used in place of `if…else` statements to complete a task. However, the program may execute more slowly in this case because there are more evaluations to be made.

### Short-Circuit Evaluation

1. Short-circuit evaluation means that the logical expression is evaluated from left to right, and as soon as the value of the entire logical expression is known, the evaluation stops.

2. The operators `&` and `|` can be used in place of `&&` and `||`, respectively, but there is no short-circuit evaluation with the operators `&` and `|`.

3. Short-circuit evaluation means that part of an expression may remain unevaluated.

### Comparing Floating-Point Numbers for Equality: A Precaution

1. When floating-point numbers are compared for equality, the results may not be as expected.

### Conditional Operator ( `?` `:` )

1. The conditional operator is a ternary operator, which means that it takes three arguments.

2. The syntax for using the conditional operator is:
   expression1 ? expression2 : expression3

3. Note that expression1 is a logical expression.

   Consider the following statements:

```
if (a >= b) {
        max = a;
}
else {
        max = b;
```

   You can use the conditional operator to simplify the writing of this if...else statement as follows:

```
max = (a >= b) ? a : b;
```

## Avoiding Bugs by Avoiding Partially Understood Concepts and Techniques

1. Even small errors can prevent a program from running correctly or from running at all.

2. Even though there are many ways to solve a problem, the approach you take must make correct use of concepts and techniques. If you have a partial understanding of a concept or technique, don't use it until your understanding is complete.

## Program Style and Form (Revisited): Indentation

1. There are two commonly used styles for placing braces. In this book, we place braces on a line by themselves. Also, matching left and right braces are in the same column, that is, they are the same number of spaces away from the left side of the program. This style of placing braces easily shows the grouping of the statements as well as matching left and right braces. You can also follow this style to place and indent braces.

2. In the second style of placing braces, the left brace need not be on a line by itself. Typically, for control structures, the left brace is placed after the last right parenthesis of the (logical) expression and the right brace is on a line by itself. This style might save some space. However, sometimes this style might not immediately show the grouping or the block of the statements.

3. No matter what style of indentation you use, you should be consistent within your programs and the indentation should show the structure of the program.

## `switch` Structures

1. Java's switch structure gives the computer the power to choose from many alternatives.

   The general syntax of a switch statement is:

```
switch (expression)
{
        case value1: {
                statements1
                break;
        }

        case value2: {
                statements2
                break;
        }
        .
        .
        case valuen: {
                statementsn
                break;
        }
        default: {
        statements
        }
}
```

2. The expression is usually an identifier, and the value is always an integral. The value of the expression determines which statement is selected for execution, and therefore the particular case values must be unique.

3. One or more statements may follow a `case` symbol, and curly braces are unnecessary to group them. Furthermore, the `break` statement may or may not appear after each statement.

4. When the value of the expression is matched against a `case` value (also called a label), the statements execute until either a break statement is found or the end of the switch structure is reached.

5. If the value of the expression does not match any of the `case` values, the statements following the default label execute. If the `switch` structure has no default label and if the value of the expression does not match any of the `case` values, the entire `switch` statement is skipped.

6. A `break` statement causes an immediate exit from the `switch` structure.

### Choosing Between an `if...else` and a `switch` Structure

1. No fixed rules exist that can be applied to decide whether to use an `if...else` structure or a `switch` structure to implement multiple selections.

2. If multiple selections involve a range of values, you should use either an `if...else` structure or a `switch` structure, wherein you convert each range to a finite set of values.

3. If the range of values is infinite and you cannot reduce them to a set containing a finite number of values, you must use the `if...else` structure.

## Comparing Strings

1. In Java, strings are compared character by character, starting with the first character and using the Unicode collating sequence.

2. The character-by-character comparison continues until one of three conditions is met: a mismatch is found, the last characters have been compared and are equal, or one string is exhausted.

3. If two strings of unequal length are compared and they are equal until the last character of the shorter string, then the shorter string is evaluated as less than the larger string.

4. The method `compareTo` in the class `String` can be used to compare objects of the class, and step through the syntax of `compareTo`.

5. The method `equals` can be used to compare two strings and determine if they are equal.

6. The expression `(str1 == str2)` determines whether the values of `str1` and `str2` are the same; that is, if `str1` and `str2` point to the same `String` object.

**Strings, the Assignment Operator, and the Operator `new`**

1. `String` variables can be initialize with and without the `new` operator.

2. When a `String` variable is initialized without the `new` operator, the computer checks whether there is already a `String` object with the given value. If so, the address of that object is stored in the variable.

3. If the `String` variable is initialized using the `new` operator, the computer creates a new `String` object with the given value, regardless of whether such a `String` object already exists.

## Additional Resources

1. Conditional statements in Java:
   http://download.oracle.com/javase/tutorial/java/nutsandbolts/op2.html

2. An article about logical implication:
   http://planetmath.org/encyclopedia/LogicalImplication.html

3. An article from Microsoft on floating point accuracy:
   http://support.microsoft.com/kb/q69333/

4. An enhanced Java switch statement:
   http://java.dzone.com/articles/java-switch-steroids

## Key Terms

- **action statement –** The statement following the logical expression.
- **associativity –** If the operators in an expression are evaluated left to right, their associativity is said to be from left to right.
- **block –** see **compound statement**
- **Boolean expression –** see **logical expression**
- **Boolean operator –** see **logical operator**
- **Boolean values –** see **logical values**
- **branch –** Altering the flow of program execution by making a selection or choice.

- **compound statement –** A structure consisting of a sequence of statements enclosed in braces, functioning as if it were a single statement.
- **condition –** An expression that evaluates to true or false and is used to determine whether to execute the statement block following it in a selection structure.
- **conditional expression –** A statement that uses a conditional operator.
- **conditional operator –** A ternary operator that has the syntax `expression1 ? expression2 : expression3`, which is equivalent to an `if…else` statement where `expression1` is the condition and if true `expression2` is executed, otherwise `expression3` is executed.
- **equality operator –** The Java symbol, which consists of two equal signs (==) and determines whether two expressions are equal.
- **logical expression –** An expression that has a value of either true or false.
- **logical operator –** Used to combine logical expressions.
- **logical value –** The values true and false.
- **loop –** Altering the flow of program execution by repetition of statement(s).
- **nested –** When control statements are located within other control statements.
- **relational operator –** An operator that allows you to make comparisons in a program.
- **repetition –** When the program repeats particular statements a certain number of times depending on one or more conditions.
- **selection –** When a program executes particular statements depending on one or more conditions.
- **sequence structure –** A control structure in which the flow of control proceeds from one statement to the next statement in a program.
- **short-circuit evaluation** (of a logical expression) **–** A process in which the computer evaluates a logical expression from left to right and stops as soon as the value of the expression is known.
- **ternary operator –** A conditional operator that takes three arguments.
- **truth table –** A table that shows the value of an expression for each possible value of its constituent Boolean variables.