Boolean Operations
&
Selection Statements

Introduction

This unit covers how to build programs that make basic decisions.

Comparison Operations

Comparison operators are operators that are used to compare two values. Comparisons result in a boolean value. The following table covers the comparison operators:

Symbol	Name	Description	Opposite
<	Less than	True when the left value is smaller than the one on right, false otherwise	>=
>	Greater than	True when the left value is larger than the one on right, false otherwise	<=
<=	Less than or equal to	True when the left value is smaller than or equal to the one on right, false otherwise	>
>=	Greater than or equal to	True when the left value is larger than or equal to the one on right, false otherwise	<
==	Equal	True when both values are equal	!=
!=	Not Equal	True when the values are different	==

Comparing Objects

Objects are not compared with the comparison operations. To compare Objects there are two methods all Objects have.

Method	Description
equals(Object o)	Returns true if the called Object is equal to the received Object.
compareTo(Object o)	Returns 0 if the calling/received Object are the same, a negative number if the calling Object is smaller and a positive number if the received Object is smaller.

Example 1:

```
String a = “Billy”;
String b = “Turtle”;

boolean comp = a.equals(b);

System.out.println(comp);
```

Output:

false

Example 2:

```
String a = “Billy”;
String b = “Cat”;

int comp = a.compareTo(b);

System.out.println(comp);
```

Output:

-1

Example 3:

```
String a = “Billy”;
String b = “Cat”;

int comp = b.compareTo(a);

System.out.println(comp);
```

Output:

1

Conjunction Operators

Conjunction operators are operators that combine two boolean values and produce a new boolean result.

AND

The AND conjunction produces a true value when both of its operands are true and false otherwise. The symbol for AND is &&.

&& Chart

Operand 1	Operand 2	Result
false	**false**	**false**
false	**true**	**false**
true	**false**	**false**
true	**true**	**true**

OR

The OR conjunction produces a true value when at least one of its operands are true and false otherwise. The symbol for OR is ||.

|| Chart

Operand 1	Operand 2	Result
false	**false**	**false**
false	**true**	**true**
true	**false**	**true**
true	**true**	**true**

XOR

The XOR conjunction produces a true value when one operand is true and one the other operand is false. When both operands are the same it will produce a false. The symbol for XOR is ^.

^ Chart

Operand 1	Operand 2	Result
false	**false**	**false**
false	**true**	**true**
true	**false**	**true**
true	**true**	**false**

Short circuit

Sometimes the result of a conjunction is known after evaluating the left operand, so the right operand skipped. This process of not evaluating the right operand in a conjunction is called **short circuiting**.

An AND short circuits when the left operand is false. An OR short circuits when the left operand is true. XOR never shot circuits, because both sides have to be known for it be evaluated.

NOT

The exclamation symbol is used for NOT. A **NOT** negates a boolean value or expression. A negated boolean becomes its opposite. A true becomes false and a false becomes true. To use the NOT operator place an exclamation before something that evaluates to a boolean value. If you have two NOTs together, they negate each other.

De Morgan's Law

When a NOT is applied to a Boolean expression it is not always easy to determine what the expression will accept. **De Morgan's Law** is a set of rules for converting an expression containing NOTs into an expression that does not contain any NOTs. This will make the expression easier to read.

Steps for applying De Morgan's Law to a NOT

1. Distribute the NOT
2. Change effected comparison operators to their opposite
3. Change effected conjunctions
    a. ANDs changes to ORs
    b. ORs changes to ANDs

Example 1:

```
! x <  3 && y > 5
x >= 3 && y > 5
```

Example 2:

```
!( x <  3 && y != 5)
! x< 3 !&& ! y != 5
x >= 3 || y == 5
```

Example 3:

```
!( !z ==7 || x++ <= 19)
!!z == 7 !|| ! x++ <= 19
z == 7 && x++ > 19
```

Note: ! cannot be applied to a conjunction, but it is useful to write the ! next to the conjunction while working through De Morgan's Law.

Grouping Statements

When programing, sometimes statements must be grouped together. Statements can be grouped together by placing an opening curly brace before the code and a closing curly brace after it.

Selection Statements

Selection statements decide whether or not to run a segment of code. When working with selection statements the term **condition** will refer to a boolean expression.

if statements

An if statement is a statement that runs the code inside it when its condition is true and skips the code when the condition is false. The code inside the if is the following line of code or the following group of code.

Format for an:

```
if(condition)
        // code
```

OR

```
if(condition)
{
        // code
}
```

Example:

```
if(x<0)
        System.out.println("Error: negative numbers are not allowed!");
```

else if

An else if statement can only be placed immediately after an if or another else if statement. Else if statements are an extension of the else ifs before it and the if that those else ifs follow. Else if statements work similar to an if, but only get evaluated when the if/else ifs before it had false conditions.

Formant for an else if:

```
if(condition)
{
        // code
}
// other possible else ifs
else if(condition)
{
        // code
}
```

Example:

```
if(time < 1200)
        System.out.println("It is morning.");
else if(time >= 1200 && time < 1800)
        System.out.println("It is the afternoon.");
else if(time >= 1800)
        System.out.println("It is night.");
```

else

An else statement can only be placed immediately after an if or else if statement. The code in an else statement gets run when the if/else ifs before it had false conditions.

Format for an else:

```
if(condition)
{
        // code
}
// possible else ifs
else
{
        // code
}
```

Example:

```
if(time < 1200)
{
        System.out.println("It is morning.");
}
else if(time >= 1200 && time < 1800)
{
        System.out.println("It is the afternoon.");
}
else
{
        System.out.println("It is night.");
}
```

Nested Selection Statements

Selection statements are allowed to be placed inside of one another. Doing this is called is called **nesting**.

Example:

```
if(classification.equals(“freshman”) == true)
{
	if(studentID <=100)
	{
		System.out.println(“Go to room 143.”);
	}
	else
	{
		System.out.println(“Go to room 211.”);
	}
}
else if(classification.equals(“sophomore”) == true)
{
	if(studentID <=100)
	{
		System.out.println(“Go to room 116.”);
	}
	else
	{
		System.out.println(“Go to room 231.”);
	}
}
```

Ifs with Semicolons

If you put a semicolon after an if it will nullify your if. When this happens the code below the if will always be executed, regardless of the how the condition evaluates. The reason the contains the empty statement (;) and the code below the if is code that follows the if with its empty statement.

Example:

```
if(90<=grade);
	System.out.pritnln(“A”);
```

How it looks to the compiler:

```
if(90<=grade)
	;
System.out.pritnln(“A”);
```

Yoda Notation

Yoda Notation is a term for listing the variable on the right of a comparison, instead of on the left.

Example:

In Normal Notation	In Yoda Notation
x==5	5==x

The reason yoda notation is desirable is the compiler will produce an error if you use a = instead of ==.

In Java normal notation will produce an error when done with an =, except when the value and the variable are both of type boolean. In this circumstances the variable will take on the boolean value and then the statement will be skipped if the variable takes on false or it will be processed if the variable takes on true.

In most languages 0 is seen as false and all other numbers are seen as true. Because of this the compiler does not catch the error if you use = in normal notation.

The reason yoda notation will cause an error on an = is literal values cannot be changed.

Because yoda notation helps prevent unintended logic errors, that are difficult to find, many companies have their programmers use yoda notation.

Ternary Operator

The ternary operator gives a value based on a boolean expression. When the expression is true the first value is used. When the expression is false the second value is used.

Format for the ternary operator:

(expression)?value1:value2;

Example:

String canDrive = (age>=16) ? ”You can drive.” : ”You cannot drive”;

Switch

Switch statements are used as a shortcut for doing multiple ifs based on the value of a single variable. Switch statements can only be written for the following data types: byte, short, int, long, char, boolean and string.

Format for a switch:

```
switch(variableName)
{
        case value1:
                // statements for when the value is 1
                break;
        case value2:
                // statements for when the value is 2
                break;
        case value3:
                // statements for when the value is 2
                break;
        default:
                // statements for when the value is not 1, 2 or 3
}
```

The switch statement will attempt to match the value of the variable with a value from a case. Once a case if found that matches the value of the variable it will execute every line in the switch until a break is reached or the switch is fully run.

The break statement is used to stop the switch from running the remaining lines.

The default is optional and works likes an else. When the variable's value does not match any of the case values the code below default will be run.

Scope

Scope defines where variables exist. A variable only exists below its declaration and in the curly braces of its creation.

Examples:

No Errors	**Error on line 8** **Error on line 10** **Y does not exist**

int x = 2;
int y = 0;
if(x<2)
{
  y = 2;
}
else
{
  y = 4;
}
System.out.println(y);
	int x = 2;
	if(x<2)
	{
	  int y = 2;
	}
	else
	{
	  y = 4;
	}
	System.out.println(y);

Random Numbers

This section will cover how to make random numbers.

Random Number from [0 to 1):

Math.random()

Random number from [0 to x):

Math.random()*x

Random number from [x to x+y):

(Math.random()*y)+x

Examples:

- [0 to 50) (double)
  - Math.random()*50
- [0 to 50] (int)
  - (int)(Math.random()*51)
- [5 to 25] (int)
  - (int)(Math.random()*21) + 5
- [-5 to 10] (int)
  - (int)(Math.random()*16) – 5

Blue Pelican Sections

Lesson 8
Lesson 9
Lesson 10

Terms

Comparison operators	Operators that compare two values and result in a boolean value.
Condition	The boolean expression for a selection or repetition statement.
Conjunction operators	Operators used to join two boolean values into one.
De Morgan's Law	A set of rules for converting an expression containing NOTs into an expression that does not contain any NOTs.
Nesting	When groups of code are placed in other groups of code.
Scope	Where are variable exists and can be accessed.
Selection statements	Statements that are used to determine whether a code segment should be executed or skipped.
Short circuit	When the right operand in a conjunction is not evaluated, because the result is already known after evaluating the left operand.