CHAPTER 7-CONDITIONAL OPERATIONS IN JAVA

7.1 Objective

- Understand conditional constructs, boolean operations and expressions, logical operators and precedence
- Perform simple, nested and complex conditional operations
- Ability to choose right conditional constructs

7.2 Course Content

7.2.1 Conditional Operations

There are many scenarios in our everyday life, where we need to take some decisions based on some criteria or condition. Decision making is one of the major part of Java also.

Conditional logic is involved when different operations are to be performed based on whether a decision is true or not. Conditional expressions consists of conditions that results in a boolean value and performs actions based on the boolean result. There are three types of decision making statements in Java. They are:

1. if – This statement is the most basic of all control flow statements. "if" statement has a certain condition expression. It executes a certain section of code only if that particular condition is true. If the condition is false, control jumps to the end of the if statement.

```
if(condition)
{
  /* statement(s) will execute if the condition is true.
        For example, a=5
        Checking condition -
              if(a>3)
                    {
                    This code will be executed because 'a' is greater than 5.
        }
*/
}
```

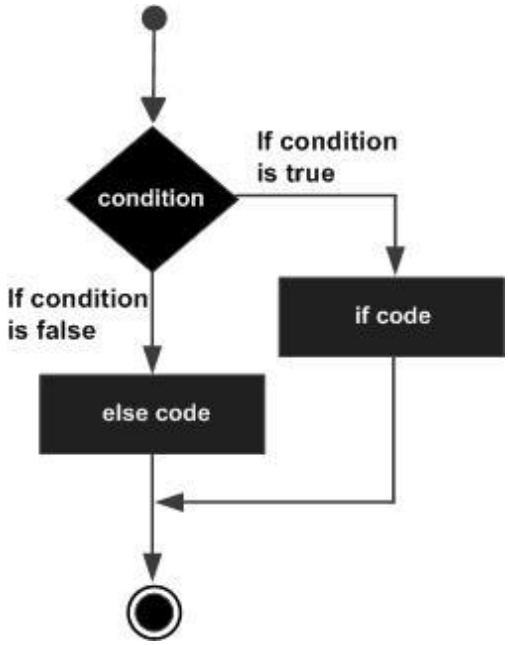
2. If ... else – This statement provides a secondary path of execution when the condition becomes false. There can be nested if then else statement also.

```
if(condition)
        {
              /* statement(s) will execute if the condition is true.*/
        }
else
        {
               /* statement(s) will execute if the boolean condition is false */
```

```
}
```

In the above example, if a=5 and condition is to check if a>6 then it will go to else block.

Flow Diagram:



3. switch – A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being given as input is checked for each case.
This statement helps to select one out of the many choices based on an integer or String value. The conditions will accept only integer, character or String. The syntax is different from if else statements. Here we are comparing the input value with different cases. We can set a default case for inputs which does not satisfy any case.
The syntax of the switch statement is as follows.

```
switch (expression) {
        case value_1 :
                        statement(s);
                        break;
        case value_2 :
                        statement(s);
                         break;
        .
        .
        .
        case value_n :
                        statement(s);
                        break;
        default:
                        statement(s);
}
```

Failure to add a break statement after a case will not generate a compile error but may have more serious consequences because the statements on the next case will be executed.

The following rules apply to a switch statement:

1. The variable used in a switch statement can only be a byte, short, int, char or String(It is allowed from jdk 1.7 version.).

2. You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.

3. The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.

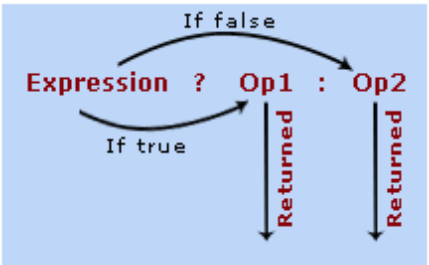
4. When the variable is found equal to a case, the statements following that case will execute until a break statement is reached. When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement. Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.

A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

7.2.2 Ternary Operator and If Else Statement

Java supports another conditional operator that is known as the ternary operator "?:" and basically is used for an if-then-else as shorthand

boolean expression ? operand1 :
operand2;
The "?:" operator evaluates an expression which may also be an operand and returns operand1 if the expression is true; otherwise returns operand2, if the expression is false. We can understand this thing with the help of a diagram shown as:



If we analyze this diagram then we find that, operand1 is returned, if the expression is true; otherwise operand2 is returned in case of false expression.
Lets have an example implementing some Logical operators:

```
class ConditionalOperator {


 public static void main(String[] args){
  int x = 5;
  int y = 10, result=0;
  boolean bl = true;
  if((x == 5) && (x < y))
  System.out.println("value of x is "+x);
  if((x == y) || (y > 1))
  System.out.println("value of y is greater than the value of x");
 result = bl ? x : y;
  System.out.println("The returned value is "+result);
 }
}
```

Output of the Program:

value of x is 5
value of y is greater than the value of x
The returned value is 5

Thus we can conclude the ternary operator can be considered as the short hand of if then else statement.

7.2.3 Conditional (Logical) Operators

Conditional operators return a true or a false value based on the state of the variables i.e. the operations using conditional operators are performed between the two boolean expressions.

Symbol	Name of the Operator
&	AND
&&	Conditional-AND
|	OR
||	Conditional-OR
!	NOT
? :	Ternary (shorthand for if-then-else statement)

I. AND (&) and Conditional-AND (&&) operators:
The AND(&)operator is similar to the Conditional-AND operator (&&).Both of its operands are of boolean type, even these operands may be boolean expressions. Other Relational operators can also be

used with these operators.

Lets use a Truth Table to know the status of an output

Op1 or Exp1	Op2 or Exp2	Result
TRUE	TRUE	TRUE
FALSE	FALSE	FALSE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE

If we analyze the table we find that result is always true only in case of first condition where both operands(or expressions) are true. On the other hand result is always false in other conditions. Note that this table works alike for & and && operators.

In case of "&" operator,if both operands or expressions are true,the result is always true, otherwise it is false if either left-hand or right-hand operand is false.

In case of "&&" operator, the result is also true, if both operands or expressions are true.
But this operator evaluates only the left-hand operand. It doesn't evaluate the right-hand operand if the left-hand operand is false then it will not jump to the right-hand operand to evaluate it, and will come out from that statement and read the next statement. That's why this mechanism is known as short-circuiting. Consider the following statements where the second expression returns false after evaluating only the left-hand operand.
True && true = true;// both operands
are evaluated
false && true = false;// only left-
operand is evaluated

But the "&" operator always evaluates both of its operands whether the first operand is true or false. Consider the following statements where the second expression returns false after evaluating the right-hand operand.

true & true = true;// both operands are
true
true & false = false;// both operands are
evaluated

II. OR (|)and Conditional-OR (||)operators: Likewise, the OR operator(|) is similar to the Conditional-OR operator (||) and returns true, if one or another of its operand is true.
Lets use a Truth Table to know the status of an output that works alike for"|" and "||" operators.

Op1 or Exp1	Op2 or Exp2	Result
TRUE	TRUE	TRUE
FALSE	FALSE	FALSE
TRUE	FALSE	TRUE

FALSE	TRUE	TRUE

If we analyze the table then we find that, result is always false only if both operands or expression are false. On the other hand, result is always true in rest of the other conditions.

Still these exists a major difference in their mode of use:
The "|" operator always evaluates both of its operands and returns true if one or other of its operand is true. Otherwise false if both the conditions are false. Consider the following statements where the second expression returns alse after evaluating the right-hand operand.

```
true | false = true; // left operand is true but
both are evaluated
false | false = false;//both operands are
evaluated
```

In case of "||" the result is also true, if one of the both operands is true. Otherwise it evaluates to falseif both operands are false. But this operator conditionally evaluates the right-hand operand only if the left-hand operands is false. Like the Conditional-AND operator, this mechanism is also known as short-circuiting. Consider the following statements where the first expression returns true after evaluating the right-hand operand.

```
false || true = true;// both operands
are evaluated
true || false = true;// only left-
operand is evaluated
```

III. NOT ("!") operator :
The NOT ("!") operator performs the boolean NOT operation on a single operand or an expression. It checks the boolean status of a current operand or expression and reverses the value of a boolean expression i.e. if the current value of an operand or expression is true then it reverses as false; but if the value of an operand or expression is false then it reverses as true.

Consider the following example:

```
class BoolNotDemo {
  public static void main(String[] args){
  int x = 2;
  int y = 1;
  boolean bl;

 bl = !(x > y); // bl is false
 System.out.println("x is not greater than y:"+bl);

 bl = !(y > x); // bl is true
 System.out.println("y is not greater than x:"+bl);
  }
}
```

Output of the Program:
x is not greater than y: false

y is not greater than x: true

7.3 Test Your Knowledge

1. What is the output of relational operators?
a) Integer
b) Boolean
c) Characters
d) Double

Ans : b

2. What is the output of this program?

```
class Relational_operator {
    public static void main(String args[])
    {
        int var1 = 5;
        int var2 = 6;
        System.out.print(var1 > var2);
    }
}
```

a) 1
b) 0
c) true
d) false
Ans: d

3. Which of these operators can skip evaluating right hand operand?
a) !
b) |
c) &

d) &&
Ans: d

4. Which of these statement is correct?
a) true and false are numeric values 1 and 0.
b) true and false are numeric values 0 and 1.
c) true is any non zero value and false is 0.
d) true and false are non numeric values.
Ans: d

5. Which of the following in switch block need not have a value to be checked
a) Case
b) Switch
c) Default
Ans: c

6. what is the output?

```
public static void main(String[] args)
   {
      int var1;
      if (var1)
      {
         System.out.println("Inside If Condition");
      }

   }
```

a) Inside If Condition
b) does not print anything
c) Compilation Error
d) Run Time Error
Ans. c

7. what is the output?

```
public static void main(String[] args)
   {
      int var1;
      if (var1==0)
      {
         System.out.println("Inside If Condition");
      }

   }
```

a) Inside If Condition
b) does not print anything
c) Compilation Error
d) Run Time Error
Ans. C