Java Operators

MODULE NAME: OBJECT ORIENTED PROGRAMMING WITH JAVA

MODULE CODE: CST 05206

INSTRUCTORS: JAMILA L.

Introduction

An operator is a special symbol that performs specific operation on operand(s), and then returns a result.

- For example: + is an operator to perform addition.
- Java has wide range of operators to perform various operations.

Introduction cont...

- Types of operators in Java:
  - Arithmetic operators
  - Relational and equality operators
  - Logical operators
  - Assignment operators
  - Concatenation operator
  - Special operators

Arithmetic operators

- Java supports various arithmetic operators for all integer and floating point operators.
- Most Java programs perform calculations using the arithmetic operators.
- The table below provides a list of Java arithmetic operators.

Operator	Use	Description
+	op1 + op2	Adds op1 and op2
-	op1 - op2	Subtracts op2 from op1
*	op1 * op2	Multiplies op1 by op2
/	op1 / op2	Divides op1 by op2
%	op1 % op2	Computes the remainder of dividing op1 by op2

The table shows list of Java arithmetic operators.

Arithmetic operators cont...

Note the use of various special symbols not used in algebra.

- The asterisk (*) indicates multiplication and the percent sign (%) denotes the remainder operator
- The arithmetic operators are all binary operators.
- E.g., the expression 3 + 7 contains the binary operator + and the operands 3 and 7.

Arithmetic operators cont…

```
//Example:

public class Product {
    public static void main(String[] args){
        int number1=2, number2=12, product;
        product=number1*number2;
        System.out.println("The product of two numbers is
"+product);
    }
}
```

Unary operators

Java also provides the unary increment operator (++), and the unary decrement operator (--)

- If a variable c is to be incremented by 1, the increment operator ++ can be used rather than the expressions c = c + 1 or c += 1.
- If increment or decrement operators are prefixed, they're referred to as the pre-increment or pre-decrement operators, respectively.
- If increment or decrement operators are post-fixed, they're referred to as the post-increment or post-decrement operators, respectively.

Unary operators cont…

Operator	Sample expression	Explanation
++	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

Unary operator cont…

```
public class Unary {
    public static void main(String[]args){
        int x=9, y;
        //prefixing unary operator
        y=++x;
        System.out.println("The value of y is "+y);
    }
}
```

Relational and Equality operators

Conditions in control statements are formed by using the equality operators and relational operators.

Operator	**Use** (Java Condition)	**Returns true if**
==	op1 == op2	op1 and op2 are equal
!=	op1 != op2	op1 and op2 are not equal
>	op1 > op2	op1 is greater than op2
>=	op1 >= op2	op1 is greater than or equal to op2
<	op1 < op2	op1 is less than op2
<=	op1 != op2	op1 is less than or equal to op2

Logical operators

- Java provides logical operators that may be used to form more complex conditions (decision making expressions) by combining simple conditions.
- The logical operators are && (logical AND), || (logical OR) and ! (logical NOT).

Operator	Use	Returns true if
&&	op1 && op2	op1 and op2 are both true, then result is true.
||	op1 || op2	either op1 or op2 is true, then result is true.
!	!op1	op is false

Assignment operators

Basic assignment operator (=) is used to assign one value to another, or to assign value to a variable, etc.

- For example, int number1 = 5; or sum = a + b; etc.
- Java provides several assignment operators for abbreviating assignment expressions.
- For example, the += operator adds the value of the expression on the right of the operator to the value of the variable on the left of the operator and stores the result in the variable on the left of the operator.
- Syntax for abbreviating assignment expressions: *variable operator = expression*;

Assignment operators cont…

Operator	Use	Equivalent to
+=	op1+=op2	op1=op1+op2
-=	op1-=op2	op1=op1-op2
*=	op1*=op2	op1=op1*op2
/=	op1/=op2	op1=op1/op2
%=	op1%=op2	op1=op1%op2

Concatenation operator (+)

Java allows String objects to be created by assembling smaller strings into larger strings.

- This is known as string concatenation.
- For example, the expression "hello " + "there" creates the String "hello there".

Special operators

Comma operator (,)

- Comma operators are used to link related expressions together.
- For example, int number1, number2;

Ternary operator (?:)

- This operator takes three operands and consists of two symbols ? and :

Array operator ([ ])

- Used to declare arrays, create arrays, and access array elements

Dot operator (.)

- Used to form qualified names
- For instance, used for method calling, total.add();

Special operators cont…

new

- Creates a new object or a new array
- Sum total= new Sum();
- float[] arrayOfFloats = new float[10];

instanceof

- Determines whether its first operand is an instance of its second operand

Control Statement

Introduction

Control of statements in java; "control" which sections of code in a program are executed.

- Specifying the order in which statements (actions) execute in a program is called program control.
- For example, we might want a set of statements to execute only if certain conditions are met, otherwise, the statements would not be executed.
- A condition is an expression that can be either true or false.

Types of control statements

- Three general types of control statements:
  i. Sequential control statement:
     - This is default ordering execution.

  ii. Selection (conditional) control statement:
     - Controls which block of code within several alternatives is executed.

  iii. Repetition (iterative) statement
     - Controls how many times a block of code is executed.
       - This is known as executing a loop.
       - Each execution of a loop is known as an iteration.

Selection Statement

The ability to control the flow of your program, letting it make decisions on what code to execute, is valuable to the programmer.

- One of the important functions of the selection statement is that it allows the program to select an action based upon the user's input.
- Java has three types of control selection statements.
  i. The if statement
  ii. The if...else statement
  iii. The switch (case) statement

I. if statement

The if statement either performs (selects) an action, if a condition is true, or skips it, if the condition is false.

- The if statement is called a single-selection statement because it selects or ignores a single action.

Syntax for if statement:

```
if(condition){
   //Statements to execute if TRUE
}
```

Write a Java application that checks whether the number entered by user is an odd.

II. if…else statement

The if...else statement performs an action if a condition is true and per-forms a different action if the condition is false.

- The if...else statement is called a double-selection statement because it selects between two different actions (or groups of actions).

Syntax for if…else statement:

```
if (condition) {
    //statements to execute if TRUE
}
else {
    //statements to execute if FALSE
}
```

Write a Java application that divides two numbers but checks whether the divisor is not zero.

Nested if…else statement

Another use of else is when there are multiple conditional statements that may all evaluate to true, yet you want only one of statement's body to execute.

- You can use "nested if...else statements" following an if statement and its body; that way, if the first statement is true, the "else if" will be ignored, but if the if statement is false, it will then check the condition for the else if statement.

Syntax for nested if...else statements is:

```
if (condition1) {
      //statements to execute if  condition1;
   }
   else if (condition2) {
    //statements to execute if  condition2;
   }
   else {
   //Statements to execute if all false;
  }
```

Nested if…else statement cont…

```
// The use of the nested if statement
    class Maximum{
          public static void main(String[] args){
              int max, x = 5, y = 8, z = 2;
              if (x > y && x > z)
                  max = x;
    else if (y > x && y > z)
                  max = y;
              else
                  max = z;
              System.out.println(max);
          }
      }//end class Maximum
```

Switch (case) statement

- The switch statement performs one of many different actions, depending on the value of an expression.
- The switch statement is called a multiple-selection statement because it selects among many different actions (or groups of actions).

Switch (case) statement cont…

- The syntax of switch case is as follows:

```
switch (variable)
   {
   case this_value:
   /* code to execute if variable== this-value */
    break;
   case that_value:
    /* code to execute if variable== that-value */
    break;
default:
   /* code to execute if variable does not equal the value following
   any of the cases */
   break;
}
```

Switch (case) statement cont…

The condition of a switch statement is a value.

- The case says that if variable has the value of whatever is after that case then do whatever follows the colon.
- The break is used to break out of the case statements.
- The default case is optional, but it is wise to include it as it handles any unexpected cases.
- It can be useful to put some kind of output to alert you to the code entering the default case if you do not expect it to.

Example:

Write a Java application that prints the name of the day if the user enters the number that corresponds to the day.

Switch (case) statement cont…

```
// example of using swich case
import java.util.Scanner;

class SwitchCase{
    public static void main(String[] args){
    Scanner input = new    Scanner(System.in);
    int day;
     System.out.println("Enter the day number");
        day = input.nextInt();
        switch (day) {
        case 1:
             System.out.println("Sunday");
             break;
```

Switch (case) statement cont…

```
//Continue
case 2:
    System.out.println("Monday");
    break;
case 3:
   System.out.println("Tuesday");
   break;
case 4:
   System.out.println("Wednesday");
   break;
case 5:
    System.out.println("Thursday");
    break;
```

Switch (case) statement cont…

```
//Continue
case 6:
    System.out.println("Friday");
    break;
case 7:
    System.out.println("Saturday");
    break;
default:
    System.out.println("Enter numbers between 1 to 7 inclusive");
    break;
     } // end switch case statement
   }  // end main
} // end class
```

Repetition (iterative) statements

Most programs involve repetition, or looping.

A loop is a group of instructions the computer executes repeatedly while some loop-continuation condition remains true.

- Loops are used to repeat a block of code.
- Being able to have your program repeatedly execute a block of code is one of the most basic but useful tasks in programming.
- Now, think about what this means: a loop lets you write a very simple statement to produce a significantly greater result simply by repetition.

One caveat!

- Before going further, you should understand the concept of Java's true and false, because it will be necessary when working with loops (the conditions are the same as with if statements).
- More notably, you must understand relational and equality operators, and logical operators.

Repetition essentials

We have two means of repetition:

i. Counter-controlled repetition
   - Counter-controlled repetition is sometimes called definite repetition because we know in advance exactly how many times the loop will be executed.

ii. Sentinel-controlled repetition
   - Sentinel-controlled repetition is sometimes called indefinite repetition because it is not known in advance how many times the loop will be executed.

Counter-controlled repetition

In a counter-controlled repetition, a ***control variable*** is required to count the number of repetitions.

Counter-controlled repetition requires:

- The name of control variable (or loop counter).
- The initial value of the control variable.
- The increment (or decrement) by which the control variable is modified each time the group of instructions is executed.
- The condition that tests the final value of the control variable (whether the loop should continue).

Types of Loops

There are three types of loops:

i. for loop,

ii. while loop, and

iii. do…while loop.

- Each of them has its specific uses.

for loop

- for loops are the most useful type. The syntax for a for loop is:

```
for ( control variable initialization; condition;
control variable update )
{
Code to execute while the condition is true
 }
```

- The ***control variable initialization*** allows you to either declare a variable and give it a value or give a value to an already existing variable.

for loop cont...

- Second, the ***condition*** tells the program that while the conditional expression is true the loop should continue to repeat itself.
- The ***control variable update*** section is the easiest way for a for loop to handle changing of the variable.
- Notice that a ***semicolon*** separates each of these sections, that is important. Also note that every single one of the sections may be empty, though the semicolons still have to be there.
- If the condition is empty, it is evaluated as true and the loop will repeat until something else stops it.

for loop example

```
public class ForCounter {
   public static void main( String[] args ) {
 /* for statement header includes initialization,
       loop-continuation condition and increment */
      for ( int counter = 1; counter <= 10; counter++ )
         System.out.printf( "%d  ", counter );
         System.out.println(); // output a newline
   } // end main
} // end class ForCounter
```

for loop example explanation

This program is a very simple example of a for loop.

- **Counter** is set to 1, while **counter** is less than or equal to 10 it calls System.out.println to display the value of the variable **counter**, and it adds 1 to **counter** until the condition is met.
- Keep in mind also that the variable is incremented after the code in the loop is run for the first time.

while loop

- **while** loops are very simple. The basic structure is:

```
 control variable initialization;
while (condition)
{
Code to execute while the condition is true;
control variable update;
}
```

- The true represents a Boolean expression which could be x == 1 or while (x != 7) (x is not equal 7). It can be any combination of Boolean statements that are legal.

while loop cont…

- Even, (while x ==5 || v == 7) which says execute the code while x equals 5 or while v equals 7. Notice that a while loop is like a stripped-down version of a for loop.
- However, an empty condition is not legal for a while loop as it is with a for loop. The following example below illustrates the use of WHILE loop:

while loop example

```
public class WhileCounter
{
   public static void main( String[] args )
   {
      int counter = 1; // declare and initialize control
 variable
      while ( counter <= 10 ) // loop-continuation
 condition
      {
         System.out.printf( "%d  ", counter );
         ++counter; // increment control variable by 1
      } // end while
      System.out.println(); // output a newline
   } // end main
} // end class WhileCounter
```

while loop example explanation

- This was another simple example, but it is longer than the above for loop.
- The easiest way to think of the loop is that when it reaches the brace at the end it jumps back up to the beginning of the loop, which checks the condition again and decides whether to repeat the block another time, or stop and move to the next statement after the block.

do...while loop

do…while loops are useful for things that want to loop at least once.

- The structure is:

```
control variable initialization;
do
{
 program statements;
control variable update;
} while ( condition );
```

- Notice that the condition is tested at the end of the block instead of the beginning, so the block will be executed at least once.

do...while loop cont...

- If the condition is true, we jump back to the beginning of the block and execute it again.
- A do...while loop is almost the same as a while loop except that the loop body is guaranteed to execute at least once.
- A while loop says "Loop while the condition is true, and execute this block of code",
- a do…while loop says "Execute this block of code, and then continue to loop while the condition is true".

do...while example

```
public class DoWhileTest {
   public static void main( String[] args ){
      int counter = 1; // initialize counter
      do {
         System.out.printf( "%d  ", counter );
         ++counter;
      } while ( counter <= 10 ); // end do...while
      System.out.println(); // outputs a newline
   } // end main
} // end class DoWhileTest
```

do...while loop example explanation

- Keep in mind that you must include a trailing semi-colon after the while in the above example.
- A common error is to forget that a do…while loop must be terminated with a semicolon (the other loops should not be terminated with a semicolon, adding to the confusion).

Note: Before writing a complete program, arrive at an algorithm which accomplishes the required end result you are seeking.

- You should also be able to choose the best loop that executes the desired output and understanding the statements that change the flow of execution we need and how code is repetitively executed.

Break and continue statements

- Two keywords that are very important to looping are break and continue. The break command will exit the most immediately surrounding loop regardless of what the conditions of the loop are.
- break is useful if we want to exit a loop under special circumstances. In case of a nested loop, only the loop in which the break statement occurs is terminated.

Break and continue statements cont…

- Format of this statement is

```
break;
```

```
class BreakDemo{
        public static void main(String[] args){
          int x;                          //counter
          //loop 10 times
          for(x=1;x<=10;x++) {        //if x is 5, terminate loop
                if(x==5){
                break;            //break loop only if x is 5
                }                       // end if
                System.out.printf("\n");
                 System.out.printf("%d\t",x);  //display value of x
            }             //end for
            System.out.printf("\n\nBroke out of loop at x==%d\n",x);
         }   //end main
    } // end class BreakDemo
```

Break and continue statements cont…

Continue statement

- Format of this statement is:

  continue;

- Continue statement causes the loop in which it occurs to continue. All the statements following the continue statement are executed.
- This is used to bypass (skip) a group of statements in a loop based upon certain criteria or conditions.

Break and continue statements cont…

```
//example using continue in for loop
class ContinueDemo{
    public static void main(String[] args)
    {
            int x;          //counter
            //loop 10 times
            for(x=1;x<=10;x++)
            {     //if x is 5, continue with next iteration of loop
              if(x==5){
              continue;             /skip 5
            }     // end if
            // program continues in the next slide
            System.out.printf("%d\t",x);    //display value of x
            }     //end for
            System.out.printf("\n\nUsed continue to skip printing value 5");
    }  //end main

} // end class ContinueDemo
```

Thank you!

for listen