|  |
| --- |
| **Department of Software Engineering**  **Mehran University of Engineering and Technology, Jamshoro** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Course: SWE121 – Object Oriented Programming** | | | |
| **Instructor** | Mr. Asmatullah | **Practical/Lab No.** | 02 |
| **Date** | 16-05-2022 | **CLOs** | CLO-3 |
| **Signature** |  | **Assessment Score** | 1 Marks |

|  |  |
| --- | --- |
| **Topic** | **Demonstrating various operators in Java** |
| **Objectives** | * To demonstrate the working of various operators such as Arithmetic, Relational, Logical and Bitwise operators. * To understand operator precedence. * To understand implicit and explicit casting and conversions. * Arithmetic Expression Evaluations and implicit numeric promotions in expressions. |

|  |
| --- |
| **Lab Discussion: Theoretical concepts and Procedural steps** |

**Tools:**  Java Development Kit, Text Pad

**Theory**

* Explain various categories of operators such as Unary, Binary and Ternary operators.
* Comprehend the concept behind operator precedence and their order of evaluation.
* Explain various types of expressions such as Arithmetic expressions, Boolean Expressions etc.
* Elementary rules to arrange variables, constants and operators to form an arithmetic expression that specifies a computation.
* **Explain various categories of operators such as Unary, Binary and Ternary operators.**

**Operators :** Are the symbols that perform operations / calculations on operands.

Three categories of operators :

* + **Unary Operators :** act or operate on one operand
  + **Binary Operators :** act or operate on two operands.
  + **Ternary Operators :** act or operate on three operands'

**Unary Operators:**

The most common value to add (or subtract) and then reassign into a variable is 1. In Java, increasing a value by 1 is called *incrementing*, and decreasing by 1 is called *decrementing*. Special operators are provided in Java to perform these actions. The increment operator (++) increases the value of the variable by 1, and the decrement operator (––) decreases it by 1. Thus, if you have a variable, Counter, and you want to increment it, you would use the following statement:

Counter++; // Start with Counter and increment it.

This statement is equivalent to the more verbose statement

Counter = Counter + 1;

which is also equivalent to the moderately verbose statement

Counter += 1;

**![Text

Description automatically generated with medium confidence]()**

**Prefixing Versus Postfixing**

Both the increment operator (++) and the decrement operator(––) come in two varieties: prefix and postfix. The prefix variety is written before the variable name (++myAge); thepostfix variety is written after the variable name (myAge++). In a simple statement, it doesn’t matter which you use, but in a complex statement when you are incrementing (or decrementing) a variable and then assigning the result to another variable, it matters very much. The prefix operator is evaluated before the assignment; the postfix is evaluated after the assignment. The semantics of prefix is this: Increment the value in the variable and then fetch or use it. The semantics of postfix is different: Fetch or use the value and then

increment the original variable. This can be confusing at first, but if x is an integer whose value is 5 and using a prefix increment operator you write

int a = ++x;

you have told the compiler to increment x (making it 6) and then fetch that value and assign it to a. Thus, a is now 6 and x is now 6. If, after doing this, you use the postfix operator to write

int b = x++;

you have now told the compiler to fetch the value in x (6) and assign it to b, and then go back and increment x. Thus, b is now 6, but x is now 7.

**Binary Operators:**

Binary operators are divided into ***FOUR*** category,

1. **Assignment operators:**

You saw the assignment operator (=) earlier. This operator causes the operand on the left side of the assignment operator to have its value changed to the value of the expression on the right side of the assignment operator. The expression x = a + b; assigns the value that is the result of adding a and b to the operand x.

1. **Arithmetic/Mathematical operators:**

A second category of operators is the Arithmetic operators. Five mathematical operators are addition (+), subtraction (–), multiplication (\*), division (/), and modulus (%).

The fifth mathematical operator might be new to you. The modulus operator (%) tells you the remainder after an integer division. To get the remainder of 21 divided by 4, you take 21 modulus 4 (21 % 4). In this case, the result is 1.

1. **Arithmetic Assignment operators:**

It is not uncommon to want to add a value to a variable and then to assign the result back into the same variable. If you have a variable myAge and you want to increase the value stored in it by two, you can write

int myAge = 5;

int temp;

temp = myAge + 2; // add 5 + 2 and put it in temp

myAge = temp; // put it back in myAge

The first two lines create the myAge variable and a temporary variable. As you can see in the third line, the value in myAge has two added to it. The resulting value is assigned to temp. In the next line, this value is then placed back into myAge, thus updating it. This method, however, is terribly convoluted and wasteful. In Java, you can put the same variable on both sides of the assignment operator; thus, the preceding becomes

myAge = myAge + 2;

which is much clearer and much better. In algebra, this expression would be meaningless, but in JAVA it is read as “add two to the value in myAge and assign the result to myAge.” Even simpler to write, but perhaps a bit harder to read is

myAge += 2;

This line is using the self-assigned addition operator (+=). The self-assigned addition operator adds the r-value to the l-value and then reassigns the result into the l-value. This operator is pronounced “plus-equals.” The statement is read “myAge plus-equals two.” If myAge had the value 24 to start, it would have 26 after this statement. Self-assigned subtraction (-=), division (/=), multiplication (\*=), and modulus (%=) operators exist as well.

**l-values and r-values**

An operand that legally can be on the left side of an assignment operator is called anl-value. That which can be on the right side is called (you guessed it) an r-value.

1. **Boolean Operators:**

There are two types of Boolean operators,

1. **Relational Operators:**

JAVA has six relational operators that compare two variables or numbers and return a Boolean value. i.e true(1) or false(0)

The outcome of these operators is a Boolean value.

**Operator Meaning**

==Equal to

> Greater than

>= Greater than or equal to

< Less than

<= Less than or equal to

!= not equal to

1. **Logical Operators:**

These operators are used to combine two or more Boolean expression.

JAVA provides logical operators to enable programmers to form more complex conditions by combining simple conditions. The logical operators are

**&& ( *AND* ),**

**|| (*logical OR* ),**

**! (*logical NOT*, also called *logical negation* ).**

**Ternary Operators:**

A decision making operator is called Conditional operator. It consist of two symbols, ( **? :** ) which operate on three operands. The operator returns one of two values depending on the result of an expression. It's effects are similar to the if-else statement.

**The basic syntax:** (condition) ? if\_true : if\_false

Which is basically the same as:

if ( condition)

if\_true;

else  
 if\_false;

* **Comprehend the concept behind operator precedence and their order of evaluation.**

**Understanding Operator Precedence**

It is important to understand that operators have a precedence, but it is not essential to memorize the precedence. Precedence is the order in which a program performs the operations in a formula. If one operator has precedence over another operator, it is evaluated first.

In the complex statement

x = 5 + 3 \* 8;

which is performed first, the addition or the multiplication? If the addition is performed first, the answer is 8 \* 8, or 64. If the multiplication is performed first, the answer is 5 + 24, or 29. The JAVA standard does not leave the order random. Rather, every operator has a precedence value, and the complete list is shown in the table below. Multiplication has higher precedence than addition; thus, the value of the expression is 29. When two mathematical operators have the same precedence, they are performed in left to- right order. Thus,

x = 5 + 3 + 8 \* 9 + 6 \* 4;

is evaluated multiplication first, left to right. Thus, 8\*9 = 72, and 6\*4 = 24. Now the expression is essentially

x = 5 + 3 + 72 + 24;

Now, the addition, left to right, is 5 + 3 = 8; 8 + 72 = 80; 80 + 24 = 104. Be careful with this. Some operators, such as assignment, are evaluated in right-to-left order! In any case, what if the precedence order doesn’t meet your needs? Consider the expression

TotalSeconds = NumMinutesToThink + NumMinutesToType \* 60

In this expression, you do not want to multiply the NumMinutesToType variable by 60 and then add it to NumMinutesToThink. You want to add the two variables to get the total number of minutes, and then you want to multiply that number by 60 to get the total seconds. You use *parentheses* to change the precedence order. Items in parentheses are evaluated at a higher precedence than any of the mathematical operators. Thus, the preceding example should be written as:

TotalSeconds = (NumMinutesToThink + NumMinutesToType) \* 60

Operator precedence table of those operators which we have discussed.

|  |  |  |
| --- | --- | --- |
| RANK | NAME | OPERATOR |
| 1 | Prefix and Postfix *(Unary)* | --, ++ |
| 2 | Multiply, Divide, Modulus *(Arithmetic)* | \*, /, % |
| 3 | Plus, Minus (*Arithmetic)* | +, - |
| 4 | Inequality *(Relational)* | <, >, <=, >= |
| 5 | Equality *(Relational)* | ==, != |
| 6 | AND *(Logical)* | && |
| 7 | OR *(Logical)* | | | |
| 8 | Conditional | ?: |
| 8 | Assignment | =, \*=, +=, /=, %=, -= |

* **Explain various types of expressions such as Arithmetic expressions, Boolean Expressions etc.**

**Expressions**

Anything that evaluates to a value is an expression in JAVA. An expression is said to *return* a value. Thus, the statement 3+2; returns the value 5, so it is an expression. All expressions are statements.

**Arithmetic Expressions:**

These expressions are those expressions in which simply calculations are being done,

Example: x = 2 + 3

a = b – 3

ans = 2\*a / b%3

**Boolean Expressions:**

These expressions are those expressions in which two values are being compared by using relational and logical operators and gives the value 1(TRUE) or 0(FALSE),

Example: 3 > 5 gives 0

If x = 2 and y = 2 x == y gives 1

* **Elementary rules to arrange variables, constants and operators to form an arithmetic expression that specifies a computation.**
* **Discuss the syntax/format for Type Conversion and Casting.**

**CASTING:**

The casting applies to data conversions specified by the programmer. Casting also called type casts or Type Conversion.

Two Types of Casting **Implicit** and **Explicit**

**Implicit Casting:** do not require any code from the programmer. They are automatically performed by the complier

This casting fits for the conversion of from data types(having small range) to the data type(having large range)

**Example** char c=‘A’;

int a=c; // implicit casting

**Explicit Casting:** requires some syntax for conversion.

This casting fits for the conversion of from data type(having small range) to the data type(having large range)

|  |
| --- |
| **Lab Tasks** |

1. Write java program that Declares 5 floating numbers and print their total sum and average.
2. Write java code that define radius for a circle and print the circumference, diameter and area for the circle.
3. Write a java program that takes an amount as runtime argument and prints number of bills and coins it will take to complete that amount. For example, if user enters Rs 5843, then the output should look like this:

Rs. 5000 1

Rs. 1000 0

Rs. 500 1

Rs. 100 3

Rs. 50 0

Rs. 20 2

Rs. 10 0

Rs. 2 1

Rs. 1 1