Course Name:

Advanced Java

Lecture 2
Topics to be covered

- Variables
- Operators

Variables -Introduction

- A **variables** can be considered as a name given to the location in memory where values are stored.
- One syntax of variable declaration

  dataType variableName;
- Do you imagine that the variable can change its value – yes, that is why it is called as variable.
- Is the following variable name is legal: GrandTotal ? Suggest good name for it?

Variable Names and Keywords

- Use only the characters 'a' through 'z', 'A' through 'Z', '0' through '9', character '_', and character '$'. A name can’t contain space character.
- Do not start with a digit. A name can be of any length.
- Upper and lower case count as <u>different</u> characters. So SUM and Sum are different names.
- A name can not be a reserved word.
- A name must not already be in use in this part of the program.
- A **reserved word** is a word which has a predefined meaning in Java. For example int, double, true, and import are reserved words.

Expressions

- An expression is a combination of constants (like 100), operators (like +), variables(section of memory) and parentheses (like "(" and ")") used to calculate a value.
- x = 1; y = 100 + x;
- This is the stuff that you know from algebra, like: (32 - y) / (x + 5)
- There are rules for this, but best rule is that an expression must look OK as algebra.
- Based on what you know about algebra, what is the value of 12 - 4/2 + 2 ?
- Spaces don't much matter.

Operators-An Introduction

- Operators are used to compute and compare values and test multiple conditions.
- You can connect two values or variables in a logical expression using operators.
- Operators also depict a relationship between two values or variables.
- An expression without operators is similar to a sentence without conjunctions.

Java provides the following types of operators

- **Arithmetic**
- **Assignment**
- **Unary**
- **Comparison**
- **Shift**
- **Bit-wise**
- **Logical**
- **Conditional**
- **New**

Arithmetic Operators

- Arithmetic operators are used to write expressions that produce numerical output.
- Assume integer variable A holds 10 and variable B holds 20 then

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0

Arithmetic Operators

- What is the value of –12 + 3
- An **arithmetic operator** is a symbol that asks for doing some arithmetic.

Operator	Meaning	Precedence
-	Unary minus	highest
+	Unary Plus	highest
*	Multiplication	middle
/	Division	middle
%	Modulus	middle
+	addition	low
-	Subtraction	low

Assignment Operators

- These operators are used to assign values to a variable.
- If a variable is not assigned a value, the variable is initialized with the default value.

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assigne value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Unary Operators

- Unary operators use only a single operand, such as x++. In this case, x is the single operand and the unary operator is +.
- Assume integer variable A holds 10 and variable B holds 20 then

Operator	Description	Example
++	Increment - Increase the value of operand by 1	B++ gives 21
--	Decrement - Decrease the value of operand by 1	B-- gives 19

Comparison Operators

- You can compare the values stored in two variables using the comparison operators.
- Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Shift Operators

- In memory, data is stored in binary format. A bit can have a value of one or zero. Eight bits form a byte.

Decimal	Binary Equivalent
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101
6	00000110
7	00000111
8	00001000

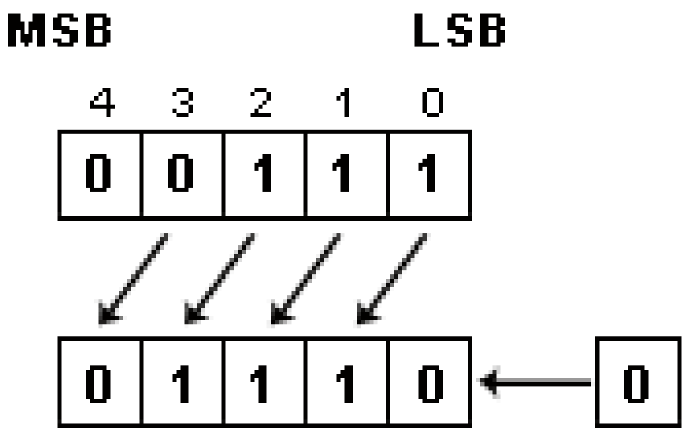
Cont….

- Shift operators work on bits of data. You use the shift operators to move the bit pattern to the left or right.

Operator	Description	Example
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>>	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

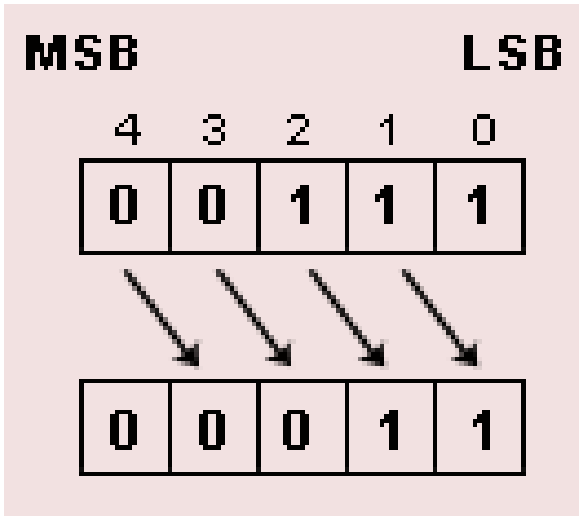
Signed Left Shift ("<<")

The signed left shift ("<<") operator shifts a bit (or bits) to the left by the distance specified in the right operand. In this case, the leftmost digit is shifted at the end of the register, and a new 0 is shifted into the rightmost position. No matter, the number is positive or negative; In both of case the leading bit position is always filled with a zero.



Signed Right Shift (">>")

The signed right shift (">>") operator shifts a bit (or bits) to the right by the distance specified in the right operand and fills the left most bit by the sign bit. In this case the rightmost bit (or bits) is shifted out, and a new 0 is filled with the sign bit into the high-order bits to the left position if the left operand is positive; otherwise 1, if the left operand is negative. This technique is known as sign extension.



Unsigned Right Shift (">>>")

- The unsigned right shift (">>>") operator behave like the signed right shift operator. i.e. it shifts a bit (or bits) to the right. But unlike ">>" operator, this operator always shifts **zeros** into the leftmost position by the distance specified in the right operand. So the result of applying the >>>operator is always positive.
- For example, the expression **"14>>>2";** shifts all bits of the number **14** to the **right** placing a **zero** to the **left** for each blank place Thus the value **1110** becomes **0011** or **3** in decimal.
- An unsigned shift operation of a negative number generally returns the result in a positive number, because any unsigned right shift operation replaces the leading sign bit with a **zero** which indicates a positive number

Bit-wise operators

- You use bit-wise operators to evaluate complex conditions, such as depicting logical circuits.
- When you use a bit-wise operator, the operands are first converted into their binary equivalents.
- Assume integer variable A holds 60 and variable B holds 13 then

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in eather operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -60 which is 1100 0011

Logical Operators

- You use logical operators to combine the results of Boolean expressions.
- Assume boolean variables A holds true and variable B holds false then

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Conditional Operators

- Conditional operators enable you to write statements that have a conditional or a selection construct.

Operator	Description	Example
(condition)?val1:val2	Evaluates to val1 if the condition returns true and val2 if the condition returns false.	X=(y>z)?y:z X is assigned the value of y if y is greater than z, else x is assigned the value of z.

New Operator

- When you create an instance of a class, which is called an object, you need to allocate memory.
- To allocate memory for an object, there are two ways:
- Declare the object and then allocate memory using new operator.

```
<class_name><object_name>;
<object_name>= new <class_name>();
```

- Declare the object and simultaneously allocate memory using the new operator.

```
<class_name><object_name>=new <class_name>();
```