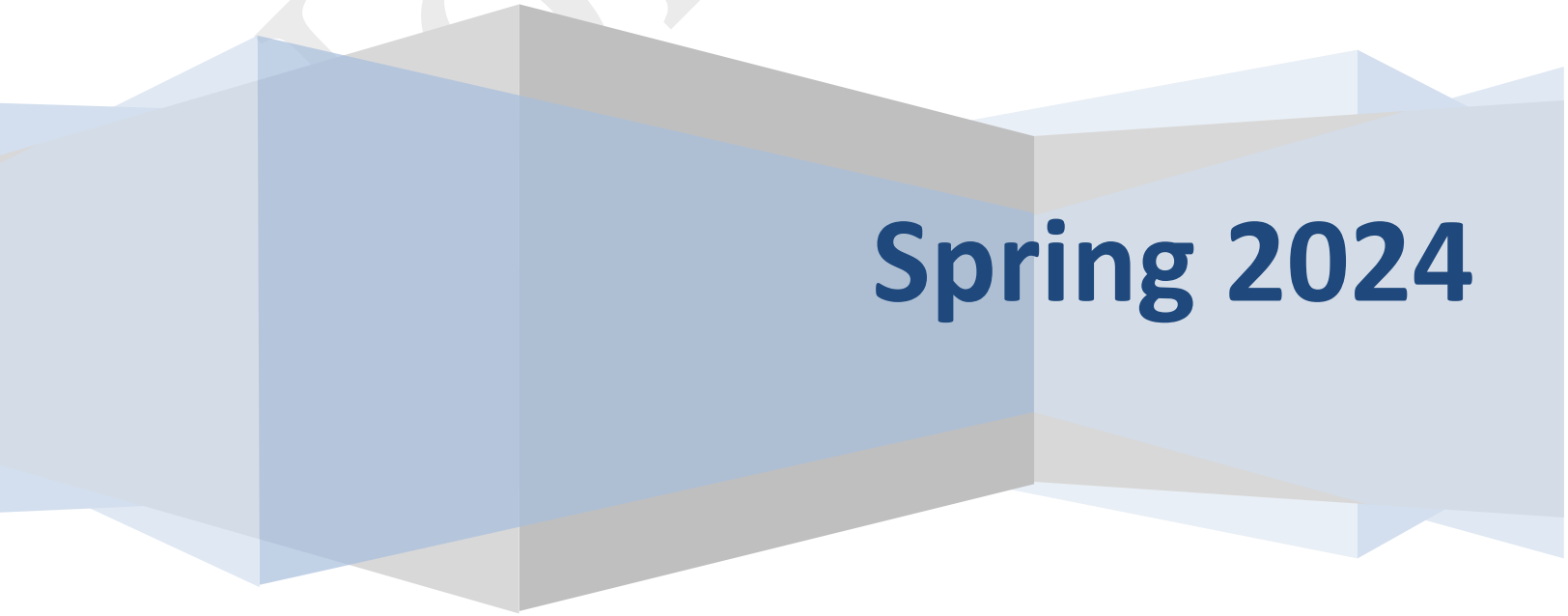
Northern University, Nowshera

Basic syntax & Operators in Java

Week # 02 - Lecture 03 – 04



Spring 2024

LEARNING OBJECTIVES:

- Previous lecture (recap)
- Basic syntax of java
- Comments in java
- Identifiers/variables
- Rules for Naming Variables in Java
- Data types
- Primitive data types
- Reference data types
- An example Java program
- Operators
- Assignment Operators
- Arithmetic Operators
- Unary Operators
- Relational Operators
- Logical Operators

BASIC SYNTAX OF JAVA

About Java programs, it is very important to keep in mind the following points.

- **Case Sensitivity** − Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
- **Class Names** − For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

Example: *class MyFirstJavaClass*

Method Names − All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

Example: *public void myMethodName()*

COMMENTS IN JAVA

Java supports single-line and multi-line comments very similar to C and C++. All characters available inside any comment are ignored by Java compiler.

Example

```
public class MyFirstJavaProgram {

  /* This is my first java program.

   * This will print 'Hello World' as the output

   * This is an example of multi-line comments.
```

```
 */
 public static void main(String []args) {
   // This is an example of single line comment
   /* This is also an example of single line comment. */
   System.out.println("Hello World");
 }
}
```

Output

Hello World

JAVA IDENTIFIERS

All Java components require names. Names used for classes, variables, and methods are called **identifiers**.

Here's an example to declare a variable in Java.

```
int speedLimit = 80;
```

Here, `speedLimit` is a variable of `int` data type, and is assigned value 80. Meaning, the `speedLimit` variable can store integer values. You will learn about Java data types in detail later in the article.

In the example, we have assigned value to the variable during declaration. However, it's not mandatory. You can declare variables without assigning the value, and later you can store the value as you wish. For example,

```
int speedLimit;
```

```
speedLimit = 80;
```

The value of a variable can be changed in the program, hence the name 'variable'. For example,

```
int speedLimit = 80;
... .. ...
speedLimit = 90;
```

- Java is a statically-typed language. It means that all variables must be declared before they can be used.
- Also, you cannot change the data type of a variable in Java within the same scope. **What is variable scope?** Don't worry about it for now. For now, just remember you cannot do something like this.

```
int speedLimit = 80;
... .. ...
float speedLimit;
```

RULES FOR NAMING VARIABLES IN JAVA

Java programming language has its own set of rules and conventions for naming variables. Here's what you need to know:

- Variables in Java are case-sensitive
- A variable's name is a sequence of Unicode letters and digits. It can begin with a letter, $ or _. However, it's convention to begin a variable name with a letter. Also,

variable name cannot use whitespace in Java.

Variable Name	Remarks
speed	Valid variable name
_speed	Valid but bad variable name
$speed	Valid but bad variable name
speed1	Valid variable name
spe ed	Invalid variable name
spe"ed	Invalid variable name

- When creating variables, choose a name that makes sense. For example, `score`, `number`, `level` makes more sense than variable name such as `s`, `n`, and `l`.
- If you choose one word variable name, use all lowercase letters. For example, it's better to use `speed` rather than `SPEED`, or `sPEED`.
- If you choose variable name having more than one word, use all lowercase letters for the first word and capitalize the first letter of each subsequent word. For example, `speedLimit`.

BASIC DATA TYPES [CLICK HERE FOR VIDEO]

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in the memory.

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

There are two data types available in Java –

- Primitve Data Types
- Reference data types

JAVA PRIMITIVE DATA TYPES

As mentioned above, Java is a statically-typed language. This means that, all variables must be declared before they can be used.

```
int speed;
```

Here, `speed` is a variable, and the data type of the variable is `int`. The `int` data type determines that the `speed` variable can only contain integers.

In simple terms, a variable's data type determines the values a variable can store. There are 8 data types predefined in Java programming language, known as primitive data types.

In addition to primitive data types, there are also referenced data types in Java (you will learn about it in later chapters).

BOOLEAN

- The `boolean` data type has two possible values, either `true` or `false`.
- Default value: `false`.

- They are usually used for true/false conditions.

Example:

```
1. class BooleanExample {
2.     public static void main(String[] args) {
3.
4.     boolean flag = true;
5.     System.out.println(flag);
6.     }
7. }
```

When you run the program, the output will be:

```
true
```

BYTE

- The `byte` data type can have values from -128 to 127 (8-bit signed two's complement integer).
- It's used instead of `int` or other integer data types to save memory if it's certain that the value of a variable will be within [-128, 127].
- Default value: 0

Example:

```
1. class ByteExample {
2.     public static void main(String[] args) {
3.
4.     byte range;
5.
6.     range = 124;
```

```
7.      System.out.println(range);
8.  
9.      // Error code below. Why?
10.             // range = 200
11.     }
12. }
```

When you run the program, the output will be:

```
124
```

SHORT

- The `short` data type can have values from -32768 to 32767 (16-bit signed two's complement integer).
- It's used instead of other integer data types to save memory if it's certain that the value of the variable will be within [-32768, 32767].
- Default value: 0

Example

```
1. class ShortExample {
2.     public static void main(String[] args) {
3.         
4.         short temperature;
5.         
6.         temperature = -200;
7.         System.out.println(temperature);
8.         
9.     }
10. }
```

When you run the program, the output will be:

```
-200
```

INT

- The `int` data type can have values from -2^{31} to $2^{31}-1$ (32-bit signed two's complement integer).
- If you are using Java 8 or later, you can use unsigned 32-bit integer with minimum value of 0 and maximum value of $2^{32}-1$.
- Default value: 0

Example:

```
1. class IntExample {
2.     public static void main(String[] args) {
3.
4.     int range = -4250000;
5.     System.out.println(range);
6.     }
7. }
```

When you run the program, the output will be:

```
-4250000
```

LONG

- The `long` data type can have values from -2^{63} to $2^{63}-1$ (64-bit signed two's complement integer).
- If you are using Java 8 or later, you can use unsigned 64-bit integer with minimum value of 0 and maximum value of $2^{64}-1$.
- Default value: 0

Example:

```
1. class LongExample {
2.     public static void main(String[] args) {
3.
4.     long range = -42332200000L;
5.     System.out.println(range);
6.     }
7. }
```

When you run the program, the output will be:

```
-42332200000
```

Notice, the use of `L` at the end of `-42332200000`. This represents that it's an integral literal of `long` type. You will learn about integral literals later.

DOUBLE

- The `double` data type is a double-precision 64-bit floating point.
- It should never be used for precise values such as currency.

- Default value: 0.0 (0.0d)

Example:

```
1. class DoubleExample {
2.     public static void main(String[] args) {
3.
4.     double number = -42.3;
5.     System.out.println(number);
6.     }
7. }
```

When you run the program, the output will be:

```
-42.3
```

FLOAT

- The `float` data type is a single-precision 32-bit floating point.
- It should never be used for precise values such as currency.
- Default value: 0.0 (0.0f)

Example:

```
1. class FloatExample {
2.     public static void main(String[] args) {
3.
4.     float number = -42.3f;
5.     System.out.println(number);
6.     }
7. }
```

When you run the program, the output will be:

```
-42.3
```

Notice that, we have used `-42.3f` instead of `-42.3`in the above program. It's because `-42.3` is a `double` literal. To tell compiler to treat `-42.3` as `float` rather than `double`, you need to use `f` or `F`.

CHAR

- It's a 16-bit Unicode character.
- The minimum value of char data type is `'\u0000'` (0). The maximum value of char data type is `'\uffff'`.
- Default value: `'\u0000'`

Example:

```
1. class CharExample {
2.     public static void main(String[] args) {
3.         
4.         char letter = '\u0051';
5.         System.out.println(letter);
6.     }
7. }
```

When you run the program, the output will be:

```
Q
```

You get the output `Q` because the Unicode value of `Q` is `'\u0051'`. Here is another example.

```
1. class CharExample {
2.     public static void main(String[] args) {
3.
4.     char letter1 = '9';
5.     System.out.println(letter1);
6.
7.     char letter2 = 65;
8.     System.out.println(letter2);
9.
10.        }
11.    }
```

When you run the program, the output will be:

```
9
A
```

When you print `letter1`, you will get 9 because `letter1` is assigned character `'9'.`

When you print `letter2`, you get A because the ASCII value of `'A'` is 65. It's because java compiler treats character as integral type.

Java also provides support for character strings via java.lang.String class. Here's how you can create a String object in Java:

```
myString = "Programming is awesome";
```

Java String is an important topic which you will learn in detail in later chapters. However, if you are not a newbie in programming and want to learn it now, visit *Java String*.

Character and String Literals

- They contain Unicode (UTF-16) characters.

- For `char` literals, single quotation is used. For example, `'a'`, `'\u0111'` etc.
- For `String` literals, double quotation is used. For example, `"programming"`, `"Java 8"`
- Java also supports a few special escape sequences. For example, `\b` (backspace), `\t` (tab), `\n` (line feed), `\f` (form feed), `\r` (carriage return), `\"` (double quote), `\'` (single quote), and `\\` (backslash).

```
1. class DoubleExample {
2.     public static void main(String[] args) {
3.
4.     char myChar = 'g';
5.     char newLine = '\n';
6.     String myString = "Java 8";
7.
8.     System.out.println(myChar);
9.     System.out.println(newLine);
10.          System.out.println(myString);
11.     }
12.  }
```

When you run the program, the output will be:

```
G
Java 8
```

REFERENCE DATA TYPES [CLICK HERE FOR VIDEO]

- Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy, etc.

- Class objects and various type of array variables come under reference datatype.
- Default value of any reference variable is null.
- A reference variable can be used to refer any object of the declared type or any compatible type.

Example: Animal animal = new Animal("giraffe");

JAVA PROGRAM TO DEMONSTRATE DATA TYPES & COMMENTS

Let us look at a simple code that will print the basics discussed above.

Example

```
public class WeekTwo {
    public static void main(String[] args) {
        System.out.println("Welcome to java Programming");
        System.out.println("We are learning Basic Syntax of Java\n\n");
        System.out.println("// This is a single line comment in Java");
        System.out.println("/* This is a multi line comment in Java */");

        System.out.println("\n\nHere we are going to declare int data type");
        int intData;
        intData=10;
        System.out.println("The value of variable 'intData' is "+ intData);

        float floatData;
        floatData=2.5f;
        System.out.println("\n\nThe value of variable 'floatData' is "+ floatData);
        System.out.println("Note we used 'f' with float value");

        double doubleData;
        doubleData=123232.45;
        System.out.println("\n\nThe value of variable 'doubleData' is "+
doubleData);
    }
}
```

Output

```
Welcome to java Programming
We are learning Basic Syntax of Java

// This is a single line comment in Java
/* This is a multi line comment in Java */

Here we are going to declare int data type
The value of variable 'intData' is 10

The value of variable 'floatData' is 2.5
Note we used 'f' with float value

The value of variable 'doubleData' is 123232.45
```

OPERATORS CLICK HERE FOR VIDEO

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups.

- Assignment Operators
- Arithmetic Operators
- Relational Operators
- Logical Operators

ASSIGNMENT OPERATOR

Assignment operators are used in Java to assign values to variables. For example,

```
int age;
age = 5;
```

The assignment operator assigns the value on its right to the variable on its left. Here, `5` is assigned to the variable `age` using = operator.

There are other assignment operators too. However, to keep things simple, we will learn other assignment operators later in this article.

Example 1: Assignment Operator

```
1.  class AssignmentOperator {
2.      public static void main(String[] args) {
3.
4.          int number1, number2;
5.
6.          // Assigning 5 to number1
7.          number1 = 5;
8.          System.out.println(number1);
```

```
9.
10.         // Assigning value of variable number2 to number1
11.         number2 = number1;
12.         System.out.println(number2);
13.     }
14. }
```

When you run the program, the output will be:

```
5
5
```

More Assignment Operators

We have only discussed about one assignment operator = in the beginning of the article. Except this operator, there are quite a few assignment operators that helps us to write cleaner code.

Operator	Example	Equivalent to
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x / 5
<<=	x <<= 5	x = x << 5
>>=	x >>= 5	x = x >> 5
&=	x &= 5	x = x & 5
^=	x ^= 5	x = x ^ 5

Operator	Example	Equivalent to
|=	x |= 5	x = x | 5

Java Assignment Operators

ARITHMETIC OPERATORS

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Operator	Meaning
+	Addition (also used for string concatenation)
-	Subtraction Operator
*	Multiplication Operator
/	Division Operator
%	Remainder Operator

Java Arithmetic Operators

Example 2: Arithmetic Operator

```
1.  class ArithmeticOperator {
2.      public static void main(String[] args) {
3.
4.          double number1 = 12.5, number2 = 3.5, result;
5.
6.          // Using addition operator
7.          result = number1 + number2;
```

```
8.        System.out.println("number1 + number2 = " + result);
9.
10.       // Using subtraction operator
11.       result = number1 - number2;
12.       System.out.println("number1 - number2 = " + result);
13.
14.       // Using multiplication operator
15.       result = number1 * number2;
16.       System.out.println("number1 * number2 = " + result);
17.
18.       // Using division operator
19.       result = number1 / number2;
20.       System.out.println("number1 / number2 = " + result);
21.
22.       // Using remainder operator
23.       result = number1 % number2;
24.       System.out.println("number1 % number2 = " + result);
25.    }
26. }
```

When you run the program, the output will be:

```
number1 + number2 = 16.0
number1 - number2 = 9.0
number1 * number2 = 43.75
number1 / number2 = 3.5714285714285716
number1 % number2 = 2.0
```

In above example, all operands used are variables. However, it's not necessary at all.

Operands used in arithmetic operators can be literals as well. For example,

```
result = number1 + 5.2;
result = 2.3 + 4.5;
number2 = number1 -2.9;
```

The `+` operator can also be used to concatenate two or more strings.

Example 3: Arithmetic Operator for Strings

```
1.  class ArithmeticOperator {
2.     public static void main(String[] args) {
3.
```

```
4.        String start, middle, end, result;
5.
6.        start = "Talk is cheap. ";
7.        middle = "Show me the code. ";
8.        end = "- Linus Torvalds";
9.
10.       result = start + middle + end;
11.       System.out.println(result);
12.   }
13. }
```

When you run the program, the output will be:

```
Talk is cheap. Show me the code. - Linus Torvalds
```

UNARY OPERATORS

Unary operator performs operation on only one operand.

Operator	Meaning
+	Unary plus (not necessary to use since numbers are positive without using it)
-	Unary minus; inverts the sign of an expression
++	Increment operator; increments value by 1
--	decrement operator; decrements value by 1
!	Logical complement operator; inverts the value of a boolean

Example 4: Unary Operator

```
1.  class UnaryOperator {
2.      public static void main(String[] args) {
3.
4.          double number = 5.2, resultNumber;
5.          boolean flag = false;
6.
7.          System.out.println("+number = " + +number);
8.          // number is equal to 5.2 here.
```

```
9.
10.          System.out.println("-number = " + -number);
11.          // number is equal to 5.2 here.
12.
13.         // ++number is equivalent to number = number + 1
14.          System.out.println("number = " + ++number);
15.         // number is equal to 6.2 here.
16.
17.         // -- number is equivalent to number = number - 1
18.          System.out.println("number = " + --number);
19.         // number is equal to 5.2 here.
20.
21.          System.out.println("!flag = " + !flag);
22.          // flag is still false.
23.      }
24.    }
```

When you run the program, the output will be:

```
+number = 5.2
-number = -5.2
number = 6.2
number = 5.2
!flag = true
```

You can also use `++` and `--` operator as both prefix and postfix in Java. The `++` operator increases value by 1 and `--` operator decreases value by 1.

```
int myInt = 5;
++myInt   // myInt becomes 6
myInt++   // myInt becomes 7
--myInt   // myInt becomes 6
myInt--   // myInt becomes 5
```

Simple enough till now. However, there is a crucial difference while using increment and decrement operator as prefix and postfix. Consider this example,

Example 5: Unary Operator

```
1.  class UnaryOperator {
```

```
2.     public static void main(String[] args) {
3.
4.         double number = 5.2;
5.
6.         System.out.println(number++);
7.         System.out.println(number);
8.
9.         System.out.println(++number);
10.        System.out.println(number);
11.    }
12. }
```

When you run the program, the output will be:

```
5.2
6.2
7.2
7.2
```

When `System.out.println(number++);` statement is executed, the original value is evaluated first. The `number` is increased only after that. That's why you are getting `5.2` as an output. Then, when `System.out.println(number);` is executed, the increased value 6.2 is displayed.

However, when `System.out.println(++number);` is executed, `number` is increased by 1 first before it's printed on the screen.

Similar is the case for decrement `--` operator.

EQUALITY AND RELATIONAL OPERATORS

The equality and relational operators determines the relationship between two operands. It checks if an operand is greater than, less than, equal to, not equal to and so on. Depending on the relationship, it results to either `true` or `false`.

Operator	Description	Example

Operator	Description	Example
==	equal to	5 == 3 is evaluated to `false`
!=	not equal to	5 != 3 is evaluated to `true`
>	greater than	5 > 3 is evaluated to `true`
<	less than	5 < 3 is evaluated to `false`
>=	greater than or equal to	5 >= 5 is evaluated to `true`
<=	less then or equal to	5 <= 5 is evaluated to `true`

Java Equality and Relational Operators

Equality and relational operators are used in decision making and loops (which will be discussed later). For now, check this simple example

Example 6: Equality and Relational Operators

```
1.  class RelationalOperator {
2.      public static void main(String[] args) {
3.
4.          int number1 = 5, number2 = 6;
5.
6.          if (number1 > number2)
7.          {
8.                  System.out.println("number1 is greater than number2.");
9.          }
10.         else
11.         {
12.                 System.out.println("number2 is greater than number1.");
13.         }
14.     }
15. }
```

When you run the program, the output will be:

```
number2 is greater than number1.
```

Here, we have used `>` operator to check if number1 is greater than number2 or not.

Since, `number2` is greater than `number1`, the expression `number1 > number2` is evaluated to `false`.

Hence, the block of code inside `else` is executed and the block of code inside `if` is skipped.

If you didn't understand the above code, don't worry. You will learn it in detail in *Java if...else* article.

For now, just remember that the equality and relational operators compares two operands and is evaluated to either `true` or `false`.

In addition to relational operators, there is also a type comparison operator `instanceof` which compares an object to a specified type. For example,

INSTANCE OF OPERATOR

Here's an example of instanceof operator.

```
1.  class instanceofOperator {
2.      public static void main(String[] args) {
3.
4.          String test = "asdf";
5.          boolean result;
6.
7.          result = test instanceof String;
8.          System.out.println(result);
9.      }
10. }
```

When you run the program, the output will be `true`. It's because `test` is the instance of `String` class.

You will learn more about `instanceof` operator works once you understand *Java Classes and Objects*.

LOGICAL OPERATORS

The logical operators `||` (conditional-OR) and `&&` (conditional-AND) operates on boolean expressions. Here's how they work.

Operator	Description	Example
||	conditional-OR; `true` if either of the boolean expression is `true`	`false || true` is evaluated to `true`
&&	conditional-AND; true if all boolean expressions are `true`	`false && true` is evaluated to false

Java Logical Operators

Example 8: Logical Operators

```
1.  class LogicalOperator {
2.      public static void main(String[] args) {
3.
4.          int number1 = 1, number2 = 2, number3 = 9;
5.          boolean result;
6.
7.          // At least one expression needs to be true for result to be true
8.          result = (number1 > number2) || (number3 > number1);
9.          // result will be true because (number1 > number2) is true
10.         System.out.println(result);
11.
12.         // All expression must be true from result to be true
13.         result = (number1 > number2) && (number3 > number1);
14.         // result will be false because      (number3 > number1) is false
15.         System.out.println(result);
16.     }
17. }
```

When you run the program, the output will be:

```
true
false
```

Logical operators are used in decision making and looping.

TERNARY OPERATOR

The conditional operator or ternary operator `?:` is shorthand for `if-then-else` statement. The syntax of conditional operator is:

```
variable = Expression ? expression1 : expression2
```

Here's how it works.

- If the Expression is true, expression1 is assigned to variable.
- If the Expression is false, expression2 is assigned to variable.

Example 9: Ternary Operator

```
1.  class ConditionalOperator {
2.      public static void main(String[] args) {
3.
4.          int februaryDays = 29;
5.          String result;
6.
7.          result =  (februaryDays == 28) ? "Not a leap year" : "Leap year";
8.          System.out.println(result);
9.      }
10. }
```

When you run the program, the output will be:

```
Leap year
```