

Day-2: Java Tokens and Types of statements

Prerequisite

- Basic of any programming language and Day-1-Java Introduction

Java Comments

Comments are optional part of program that are written by developer to specify objective of a program or module in simple human friendly language. Comments are transparent to java compiler means compiler simply ignores all comments written in a code.

java supports three styles of comments that are as follow

1. **Single line comment:** Such type of comment are used when our comment have length limited to just one line. To insert a single line comment, simply start comment with two consecutive slashes (/).

Example

```
//This is a single line comment.
```

2. **Multi line comment:** Such type of comment are used when our comment have length more than one line. To insert a multi-line comment, simply start comment with a slash (/) and asterisk (*), and after completing comment just place an asterisk (*) sign and slash (/)

Example

```
/* I am multi line comment.  
   Comments are transparent to compiler  
*/
```

3. **javadocs:** They are used to generate the HTML pages of API documentation from Java source files. Separate syntaxes are used to write javadoc for classes, methods, packages etc. We will example of such comments after making a significant progress in the java.

Tokens

- It refer to smallest lexical unit of a languages
- java tokens are classified into 5 categories as follow



Identifier

It refers to sequence of one or more character that is used to identify an entity like a memory location (i.e. variable) or to a block of code (i.e. method, class etc.)

An Analogy: Look at the following laddu-box containing the laddu in itself. It is easy to observe that the laddu box is a container for laddu.



The same is when we talk about the storing the value in a variable, you can say that the variable is a

container for the value.

To create a variable syntax is...

```
data-type identifier;
```

or

```
data-type identifier = value;
```

We have a certain set of rules for picking identifier name in java-

- Can be made from [A-Z], [a-z], [0-9], _ , \$
- Can't be started with [0-9]
- Can't be a reserved keyword
- *Convention (not a rule): Variable name should not be started with _ or \$ yet they are allowed. It is good to start them with a letter.*

Tip: Identifier for methods & variables should be created according to the convention that we have already discussed, by doing so we are making the code easy to maintain for a large team, yet not following the convention will cause no compile time or run time effect but a huge chaos in maintenance and scalability.

Best variable names: principleAmount, priority, profitPercentage, ageOfApplicant, dateOfJoining

Correct variable names: gold_9, basic_salary, salary123, a, PrincipleAmount etc.

Incorrect variable names: 9gold, basic#salary, salary-123, 6, principle amount etc.

Data type

An Analogy: Look at the two types of container for the coconut oil that are commonly found in our homes.



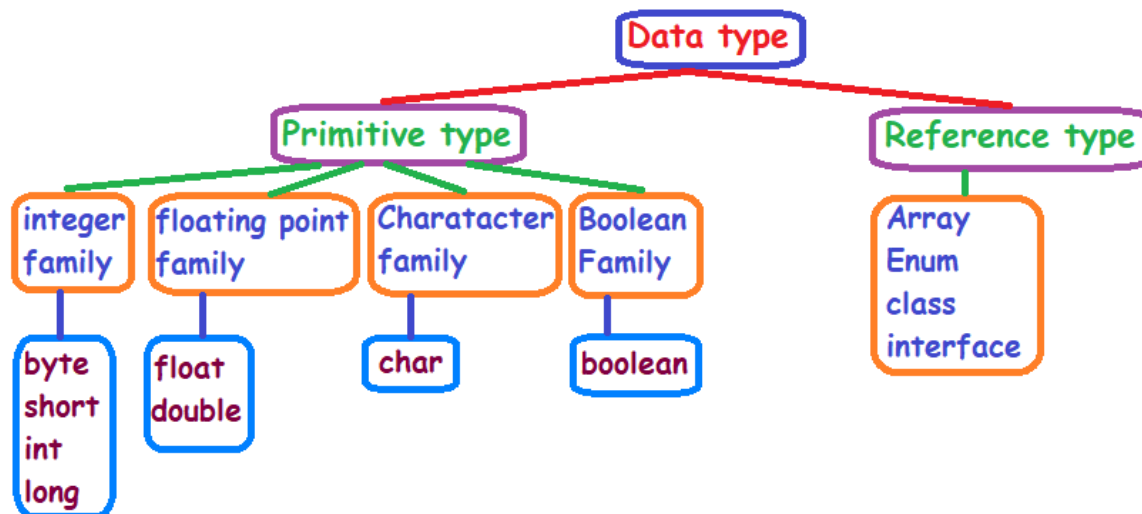
The first container is for the winter season when the coconut oil is available in the solid/semi-solid form but the second container is for the summer season when the coconut oil is available in the liquid form. Container is designed according to the state of material that they are going to have.

Data-type also does the same job, they specify the type of variable according the type value it is going to contain i.e. we have separate data types for integer numbers, fractional numbers, character value, String values etc.

A variable's data type defines following properties about a variable-

1. Values it may contain
2. Operations that may be performed on it.

Look at the following picture for classification of the data types



A primitive type is predefined by the language and is named by a reserved keyword.

Family	Data-type	Size	Range	Example
Integer family	byte	1 byte/8 bits	-128 to 127	byte c = 1;
	short	2 bytes/16 bits	-32768 to 32767	short t = 150;
	int	4 bytes/32 bits	-2147483648 to 2147483647	int j = 150000;
	long	8 bytes/64 bits	-9223372036854775808 to 9223372036854775807	long l = 2147483647L;
Floating point family	float	4 bytes/32 bits	1.4e-045f to 3.4e+038f	float f = 1.5f;
	double	8 bytes/64 bits	4.9e-324 to 1.8e+308	double d = 1.5;
Character family	char	2 bytes/16 bits	0 to 65535	char ch = 'A';
Boolean family	boolean	-	{true, false }	boolean b = false;

- There is no unsigned type specifier for integer data types in Java.
- By default all the non-fractional numerical values belong to integer type. if a value is outside the range of integer than the value must be appended with L (Capital Lion) or l (Small lion)
- By default all the fractional numerical values belong to double type. To make the value belong to the float type the value must be appended with F or f. The double value may be appended with D or d yet it makes no difference.
- The float data type is a single-precision because it can store value up to 7 decimal places and the double data type is a double-precision because it can store value up to 16 decimal places.
- The character data type uses Unicode encoding that is compatible with ASCII encoding. The Unicode is used by java for the support of internationalization.

Tip: java is an object oriented language. To support the same theme java provides object implementation of the primitive data type using Wrapper classes. Look at the table

Primitive type	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean

Literals

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation.

Family	Way of writing	Description	Example
Integer Literal	Conventional	Using [0-9], No prefix required	2758, -25, 45
	Binary	Using [0-1], Use prefix: 0b/0B	0b00001010, 0b11110101
	Octal	Using [0-7], Use prefix: 0	074, 025
	Hexadecimal	Using [0-F], Use prefix: 0x/0X	0xA1, 0X42A
Floating point Literal	Conventional	Any number of digits before & after decimal point	423.25, 25863.25
	Scientific	One digit before decimal point	4.2325E2, 2.586325E4
Character Literal	Conventional	Any characters inside "	'A', 'a'
	Unicode	Using [0-F], Use prefix: \u	'\u0041', '\u0915'
Boolean Literal	Conventional	Just write both in small letters	true, false
String Literal	Conventional	Anything inside "" is String literal	"Java", "Apple", "123_+-%"

You can place underscore character between number for the sake of improving readability. For example if you have a large number then separate them by group of three or more two improve readability of numbers.

```
long creditCardNumber = 1234_5678_9012_3456L;  
float pi = 3.14_15F;  
byte nibbles = 0b0010_0101;
```

Type conversion and casting

Automatic type conversions/widening conversion

When we assign one type of data to another type of variable then automatic type conversion take place in the following conditions met

1. The two types are compatible.
2. The destination type is larger than the source type.

Widening conversion is take place in the following cases:-

Source	Destination
byte	short, int, long, float, or double
short	int, long, float, or double
int	long, float, or double
long	float, or double
char	int, long, float, or double
float	double

Narrowing conversion & Truncation

When source type is larger than the destination type or when variable is incompatible with the data to be assigned then you have to explicitly make the value narrower so that it can fit into the target type.

To perform conversion between two incompatible types, you must use a cast. A cast is simply an explicit type conversion. It has this general form:

(target-type) value

Here, target-type specifies the desired type to convert the specified value to

When a floating-point value is assigned to an integer type, the fractional component is lost, because integers do not have fractional components. Such type of conversion is called **truncation**.

```
int i = 100, j = 200;
//short s = i it's an error because short is smaller than int
short s = (short) i;
byte b = (byte) j;
System.out.println("i is " + i);
System.out.println("s is " + s);
System.out.println("b is " + b);
```

output

```
i is 100
s is 100
b is -56
```

Another Example

```
double d = 330.14;
int i = (int) d;
System.out.println("d and i " + d + " " + i);

byte b = (byte) d;
System.out.println("d and b " + d + " " + b);
```

Output

```
d and i 330.14 330
d and b 330.14 74
```

We have found one speciality of java that is whenever you are assigning a value to a variable, java performs compatibility of assigned value to the target type and generate proper error message if found them incompatible. We can say java avoid accidental assignment of mismatching data to variable, that is why java is known as **strongly typed language**.

Some Rules about variables

Rule-1: The variables declared in the outer scope will be visible within the inner scope. However, the reverse is not true.

```
public static void main(String a[]){
    int x = 10;
    if(x == 10){
        int y = 1000;
        System.out.println("x is " + x + " and y is " + y);
    }
    //y = 10;
    System.out.println("x is " + x);
}
```

As per the discussion x is visible inside nested block while y is not accessible outside its block.

Rule-2: Make sure to create a variable before using the same. If you are creating a variable after using it, compiler will report an error.

```
System.out.println("x is " + x); // error
```

```
int x = 10;
```

Rule-3: You cannot declare a variable to have the same name as one in an outer scope.

```
public static void main(String a[]){  
    int x = 10;  
    {  
        //creating new block  
        int x = 20;//error, x already defined  
    }  
}
```

Rule-4: In java, local variable do not have any default value i.e. you have to assign a value before using them.

```
int x;  
System.out.println("x is " + x);// error
```

Keyword

A. It refer to words whose meaning is already defined to the compiler

B. Keyword cannot be used an identifier

C. Java has total 50 keywords

switch	new	for	continue	abstract
synchronized	package	goto (not used)	default	assert
this	private	if	do	boolean
throw	protected	implements	double	break
throws	public	import	else	byte
transient	return	instanceof	enum	case
try	short	int	extends	catch
void	static	interface	final	char
volatile	strictfp	long	finally	class
while	super	native	float	const (not used)

Operators

- They are the symbols that are used to specify an operation
- The values/variables on which operations applied are called operands
- A valid combination of operators and operands is called expression

Classification of operators by number of operands

Type	Description	Example
Unary	Applied on one operands only	++, --, !, ~ etc.
Binary	Applied on two operands	*, %, /, >, < etc.
Ternary	Applied on three operands	?:


Classification of operators by nature of operation

1. Arithmetic
2. Relational

3. Bitwise
4. Boolean logical
5. Conjunction/logical operator
6. Conditional
7. Assignment operators

Before talking about the details of this classification, it will be good to discuss about the rules regarding evaluation of expression in java take a look at the rules one by one

- A. No BODMAS, operators are evaluated according to priority, if operators of same priority are there then in expression then the tie will be resolved by associativity (It defines the direction of evaluation which may be left to right or right to left). Look at the priority table in java

Operators		Priority
Brackets	() []	Highest
Postfix	expr++ expr--	
Unary [@]	++expr --expr ~ ! -expr +expr	
Multiplicative	/ % *	
Additive	+ -	
Shift	>> << >>>	
Relational	< > <= >= instanceof	
Equality	== !=	
Bitwise AND	&	
Bitwise XOR	^	
Bitwise OR		
Logical AND	&&	
Logical OR		
Ternary	?:	
Assignment [@]	= += -= /= *= >>= <<= >>>=	

The operators superscripted with @ have associativity from Right to left, rest all operators have associativity from left to right.

```
int a = 5;
int b = (--a + --a);    //7
int c = (++a - a--);    //0
int d = (--a + a--);    //4
int e = (++a + a++);    //4
int f = b + c + d + e
System.out.println(f);
```

- B. Java automatically converts short and byte operand to int whenever evaluating an expression.

An Example

```
byte a = 20;
byte b = 2;
//byte c = a/b; Compile Time Error
byte c = (byte) (a/b);
System.out.println("c is " + c);
```

- C. If an expression involves several data type then final result of expression will belong to largest data type.

An Example

```
byte b = 2;
short s = 5;
int i = 10;
float f = 11.2f;
double d = 111.23;
double result = ((b*f)/i) + f*s + d;
System.out.println(result); //169.47000167846682
```

Arithmetic Operators

+, -, *, /, %, ++, --

An Example

```
System.out.println(10 + 20); //30
System.out.println(20 - 10); //10
System.out.println(20 * 10); //200
System.out.println(20 / 10); //2
System.out.println(15 / 10); //1
System.out.println(15 / 10.0); //1.5
System.out.println(15 % 10); //5
```

```
int a = 10;
int b = a++;
int c = --a;
int d = a--;
int e = ++a;
System.out.println(a + " " + b + " " + c + " " + d + " " + e);
```

Relational Operators

>, <, >=, <=, ==, !=

- Only numeric & **character type** can be compared using rest of the relational operators. (i.e. <, <=, >, >=) but all data types in java can be compared using equality test (==) and inequality (!=) test.
- Output of the relational expression is always a boolean value.

An Example

```
System.out.println(10 > 20); //false
System.out.println(10 < 20); //true
System.out.println(10 <= 10); //true
System.out.println(10 >= 9); //true
System.out.println(10 != 10); //false
System.out.println(10 == 10); //true
```

Bitwise Operators

~, &, |, ^, <<, >>, >>>

A	B	A B	A&B	A^B	~A
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

An Example

```
byte a = 9;          //Binary presentation for 9 is 0000 1001
byte b = 14;         //Binary presentation for 14 is 0000 1110
System.out.println(~a);    //-10
System.out.println(a & b); //8
System.out.println(a | b); //15
System.out.println(a ^ b); //7
System.out.println(a << 1); //18
System.out.println(a >> 1); //4

byte c = -8;
System.out.println(c >> 1); //-4
System.out.println(c >>> 1);    //2147483644
```

Boolean Logical Operators

!, &, |, ^

- Applicable for Boolean data types only
- Look at the following table for the behaviour of the same

A	B	A B	A&B	A^B	!A
false	false	false	false	false	true
false	true	true	false	true	true
true	false	true	false	true	false
true	true	true	true	false	false

An Example

```
boolean a = true;
boolean b = false;
System.out.println(!a);    //false
System.out.println(a|b);   //true
System.out.println(a&b);   //false
System.out.println(a^b);   //true
```

Conjunction/Logical Operators

&&, ||

- for &&, if left hand side operand is false then result of right hand side does not have any effect on final result so it will not be evaluated
- for ||, if left hand side operand is true then result of right hand side does not have any effect on final result so it will not be evaluated

An Example

```
byte b = 0;
boolean bln = ++b == 0 && b++ == 1;
System.out.println(b + " " + bln);    //1 false

bln = ++b == 2 || b++ == 1;
System.out.println(b + " " + bln);    //2 true
```

Conditional operator

Ternary operator is used as alternative of if then else. general form of ternary operator is

```
expression-1 ? expression-2 : expression-3;
```

Here expression1 can be any expression evaluated to a boolean value. If expression-1 is evaluated to true then expression-2 is evaluated otherwise expression-3 is evaluated.

An Example

```
int i = 1;
int j = (i == 0 ? i : i+10);
System.out.println(j);          //11
```

Assignment operator

=, +=, -=, *=, /=, &=, |= etc.

An Example

```
int a = 10;
a+=5;
System.out.println(a);
a-=10;
System.out.println(a);
a|=11;
System.out.println(a);
```

Special Symbols

[] () {}, ; * =

These special symbols are used in Java having some special meaning so can't be used for some other purpose.

Control Statements

if statement

- It tells your program to execute a certain section of code only if a particular condition evaluates to true.
- General form of if statement is:

```
if(condition){
    Group of statements (to be executed if condition is true);
}
```

An Example:

```
int i = 20;
if(i%2 == 0){
    System.out.println(i + " is an even number");
}
```

Output of program given above is

20 is an even number

Nested if statement

- One if statement can be placed inside another if statement. if that are inside another if are called nested if. Programmer can make any number of nested if as per his requirement.

An Example:

```
int age = 80;
if(age >= 18){
    System.out.println("You are not minor");
    if(age >= 65){
```

```

        System.out.println("Even you are a senior citizen");
    }
}

```

Output of the above program is

```

You are not minor
Even you are a senior citizen

```

if-else statement

- The if-else statement provides another path of execution when if clause evaluates to false.
- General form of if-else is

```

if(condition){
    Group of statements (to be executed if condition is true);
}else{
    Group of statements (to be executed if condition is false);
}

```

An Example

```

int age = 15;
if(age >= 18){
    System.out.println("You are not minor");
    System.out.println("You are free to take your decision");
}else{
    System.out.println("You are minor");
    System.out.println("Be careful! You need guidance");
}

```

Output of the above program is

```

You are minor
Be careful! You need guidance

```

if-else-if ladder

- General form of if-else-if ladder is:

```

if(condition){
    group of statements;
}else if(condition){
    group of statements;
}else if(condition){
    group of statements;
}
.
.
else{
    group of statements;
}

```

An Example

```

int score = 76;
if (score >= 60) {
    System.out.println("Hurray! first division");
} else if (score >= 48) {
    System.out.println("Second division! its okay");
} else if (score >= 36) {
    System.out.println("Thank god! third division");
} else {

```

```
        System.out.println("Alas! Repeat");
    }
```

Output of the program given above is:

Hurray! first division

switch-case statement

- It is a multi-way branching statement. It provides facility to execute different part of code based on value of expression. It is used as an alternative of large if-else-if ladder.

- General form of switch statement is:

```
switch(expression){
case value1:
    statement sequence for value1
    break;
case value2:
    statement sequence for value2
    break;
case value3:
    statement sequence for value2
    break;
.
.
case valueN:
    statement sequence for valueN
    break;
default:
    default statement sequence
}
```

An Example

```
int division = 1;
switch(division){
    case 1:
        System.out.println("Hurray! first division");
        break;
    case 2:
        System.out.println("Second division. it's okay");
        break;
    case 3:
        System.out.println("Third division. just pass");
        break;
    default:
        System.out.println("Alas! repeat");
}
```

Output of the above program is:

Hurray! first division

Comparison Between if-else and Switch-case

- Switch can only check for equality while if-else can evaluate any type of boolean expression.
- Switch cannot be used for float & boolean literals while if can be used for float literals.
- Switch case are often faster than equivalent if-else-if ladder because compiler maintains a table of all cases called "jump table". Whenever expression is evaluated its value is checked for equality with the

entries in jump table and corresponding statement sequence is executed. Thus switch statement is much faster than equivalent if-else-if ladder.

while loop

- It is most basic loop statement supported by java. while loop repeat a block of statement while its controlling condition is true.
- General form of while loop is:

```
while(condition){  
    Sequence of statements  
}
```

An Example:

```
int loopCounter = 1;  
while(loopCounter <= 10){  
    System.out.println("Loop counter is " + loopCounter);  
    loopCounter++;  
}  
System.out.println ("I am after loop");
```

Output of program given above is

```
Loop counter is 1  
Loop counter is 2  
Loop counter is 3  
Loop counter is 4  
Loop counter is 5  
Loop counter is 6  
Loop counter is 7  
Loop counter is 8  
Loop counter is 9  
Loop counter is 10  
I am after loop
```

do-while loop

- do-while loop executes body of loop first then check condition. if condition is evaluated to true then body of loop is executed again else control goes to instruction that comes next to while(condition) statement i.e. minimum one iteration is guaranteed
- General form of do-while loop is:

```
do{  
    Sequence of statement  
}while(condition);
```

An Example

```
int loopCounter = 11;  
do{  
    System.out.println("loopCounter is " + loopCounter);  
}while(loopCounter <= 10);  
System.out.println("I am after loop");
```

Output of the program above is

```
loopCounter is 11  
I am after loop
```

for loop

- for loop provides a compact way of iteration. It is a powerful and flexible construct. general form of for loop is:
for(initialization; condition; updation){
 Sequence of statement
}

An Example:

```
for(int loopCounter = 1; loopCounter <= 10; loopCounter++){  
    System.out.println("loopCounter is " + loopCounter);  
}  
System.out.println("I am after loop");
```

Output of the above program is

```
loopCounter is 1  
loopCounter is 2  
loopCounter is 3  
loopCounter is 4  
loopCounter is 5  
loopCounter is 6  
loopCounter is 7  
loopCounter is 8  
loopCounter is 9  
loopCounter is 10  
I am after loop
```

break

Unlabeled break: Normally we are using break in unlabeled form. We have already used it with switch case to prevent execution of subsequent cases. It is also used for breaking loop.

An Example

```
for(int i = 0; i <= 10; i++){  
    if(i == 5){  
        break;  
    }  
    System.out.println("i is " + i);  
}  
System.out.println("Loop completed");
```

Output of program given above is

```
i is 0  
i is 1  
i is 2  
i is 3  
i is 4  
Loop completed
```

Labeled break: Labeled breaks are used to break deep structures and mainly used to break inner loops. General form of labeled break is
break label;

An Example

```
A: {
```

```

B:{
    C:{
        for(int i = 0; i <= 5; i++){
            if(i == 2){
                break B;
            }
            System.out.println("i is " + i);
        } //closing for loop
        System.out.println("I won't execute");
    } //closing C block
    System.out.println("I won't execute");
} //closing B block
System.out.println("After block having label B");
} //closing A block

```

Output of the program given above

```

i is 0
i is 1
After block having label B

```

continue

Conventional: It brings control at the start of block, opposed to break that brings control at the end of block.

An Example:

```

for(int i = 0; i <= 10; i++){
    if(i%2 == 0){
        continue;
    }
    System.out.println("i is " + i);
}

```

Output of program given above is:

```

i is 1
i is 3
i is 5
i is 7
i is 9

```

- **Loop Label:** It is a label that has a loop, a loop label does not enclose its loop in curly braces.

```

A:for(int i = 1; i <= 3; i++){
    System.out.println("i is " + i);
    B:for(int j = 1; j<= 2; j++){
        System.out.println("j is " + j);
        if(j%2 == 0){
            continue A;
        }
    }
}

```

Output of the program given above is

```

i is 1
j is 1

```

```
j is 2
i is 2
j is 1
j is 2
i is 3
j is 1
j is 2
```

You Activity

Q.1 Match the following

- | | |
|----------------|-----------------------------|
| (a) static | (i) Character Literal |
| (b) costPrice | (ii) Keyword |
| (c) sell-price | (iii) Invalid Variable Name |
| (d) '\u8762' | (iv) Valid Variable Name |

Q.2 What is output of following code?

```
int a = 10;
boolean b = !false && (++a == 11 && a++ == 11);
System.out.println(a + " " + b);
```

- (a) 11 true
- (b) 12 false
- (c) 12 true
- (d) None of these

Q.3 Say a person want to vote in india such that two conditions are required to met

- (i) age of person must be more than or equal to 18
- (ii) person must be indian

Select the appropriate operators for the condition of if statement

```
int age = some-value;
String nationality = some-value;
if(age __ 18 __ nationality __ "indian")
    System.out.println("Can Vote");
else
    System.out.println("Can't Vote");
```

- (a) >, &&, =
- (b) >, ||, ==
- (c) >=, ||, =
- (d) >=, &&, ==

Q.4 What is output of following code?

```
int a = 5;
while(a--){
    System.out.print("* ");
}
```

- (a) * * * * *
- (b) * * * *
- (c) This loop will run for infinite time
- (d) Compile Time Error