JAVA

What you learn ?

Java Tokens & Block	
➢ Java Tokens	➢ CamelCase in java naming conventions
➢ Java Statements ,Expression & Block	

Java Tokens

- Tokens are the various elements in the java program that are identified by **Java compiler**. A token is the smallest individual element (unit) in a program that is meaningful to the compiler.

Java language contains five types of tokens that are as follows:

- Identifiers
- Keywords
- Literals
- Operators
- Special Symbols

Tokens

Java Identifiers

- ***Identifiers refer to the names of variable,class ,object ,method etc. created by the programmer***

- An identifier is a long sequence of letters(a-z & A-Z) and numbers(0-9).
- No special character except underscore (_) can be used as an identifier.
- Keyword should not be used as an identifier name.
- Java is case sensitive. So using case is significant.
- First character of an identifier can be letter, underscore (_) but not digit.

CamelCase in java naming conventions

- Java follows camel-case syntax for naming the class, interface, method, and variable.
- If the name is combined with two words, the second word will start with uppercase letter always such as actionPerformed(), firstName, ActionEvent, ActionListener, etc.
- Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.
- These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.
- By using standard Java naming conventions, you make your code easier to read for yourself and other programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code does.

key rules that must be followed by every identifier:

- **Class**
  - It should start with the uppercase letter.
  - It should be a noun such as Color, Button, System, Thread, etc.
  - Use appropriate words, instead of acronyms.
- **Method**
  - It should start with lowercase letter.
  - It should be a verb such as main(), print(), println().
  - If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().

key rules that must be followed by every identifier:

- **Variable**

- It should start with a lowercase letter such as id, name.
- It should not start with the special characters like & (ampersand), $ (dollar), _ (underscore).
- If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.

- **Package**

- It should be a lowercase letter such as java.lang
- If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.

Tokens

Keywords

Keywords in Java are predefined or reserved words that have special meaning to the Java compiler. - 50 keyword in Java

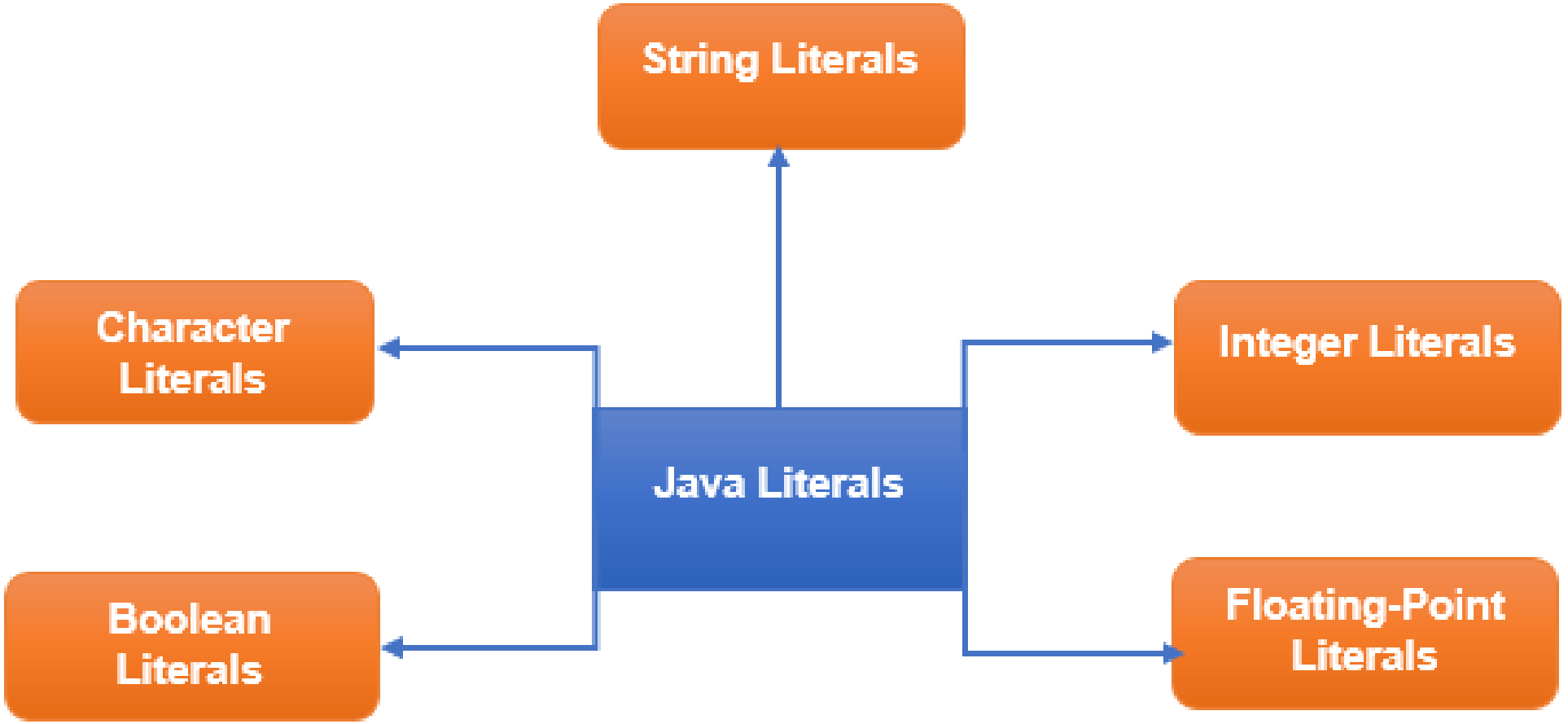
abstract	default	implements	protected	throw
assert	do	import	public	throws
boolean	double	instanceof	return	transient
break	else	int	short	try
byte	extends	interface	static	void
case	final	long	strictfp	volatile
catch	finally	native	super	while
char	float	new	switch	_
class	for	package	synchronized	_
continue	if	private	this	_

Tokens

Literals

Literals are syntactically representation of fixed value from the source

Java allows 5 kinds of literals which are:



Tokens

Integer Literals

Integer literals are basically a number sequence that doesn't contain any decimal point in between them.

1. Decimal Integer: They are integers having a **base value of 10**; i.e; containing values between **0 to 9.** It can be a positive value(+) or a negative value(-) but it cannot contain any point in between them. **int decimal_int=1234**;

2. Octal Integer: They are integers having a **base value of 8**; i.e; containing values between **0 to 7**. All octal numbers must **start with a 0**. Example: 012, 077,075, etc. **int octal_int=077;**

3. Hexadecimal Integer: They are integers having a **base value of 16**; i.e; containing values between 0 to 15. It is a special type of integer that contains both digits as well as characters. The digits range from **0 to 9** and the numbers 10 to 15 are replaced by characters a to f. Any integer starting with **0x or 0X** is considered to be a hexadecimal integer. Example: 0xff, 0x2a, 0xf1f2, etc. **int hexadec_int=0x1ff2;**

4. Binary Integer: They are integers with a **base value of 2**; i.e; contains only two digits 0 and 1. Binary integers start with a **0b** indicating that it is a binary digit. Example: 0b100101, 0b1010101, etc. **int binary_int=0b1010101;**

Tokens

Floating Point Literals

Floating-point literals are values that contain a decimal point in between them. Floating-point literals are generally double data type by default. We can assign them to float data types by adding an f or F at the end of the value.

- Floating-point Literals can also have an exponent part. This exponential part can be declared as follow: 123E-45f

- Few points to remember while declaring floating-point literals are:
  1. If no suffix is present, the default data type is double.
  2. F or f suffix represents a floating data type.
  3. D or d suffix represents a double data type.

- **float val_float=1.7732f; float val_float1=1.7732F;**

- **float val_double=1.7732d;**

- **double val_double1=1.7732;**

Tokens

Character Literals

Character Literals in java are represented with a single quote.

- **char charVar='A';**
- **char uniChar='\u0061';** // Char literal can specify in unicode representation : \u0061 here 0061 represent hexadecimal number
- **char escChar='\n';**

Tokens Escape Sequence Character Literals

Escape Sequence	Functionality	Escape Sequence	Functionality
\n	Used to insert a new line.	\?	Used to add a Question mark
\t	Used to insert a horizontal tab.	\on	Used to represent an octal number
\b	Used to insert a blank space.	\xHn	Used to represent Hexadecimal number
\v	Used to insert a Vertical tab.	\uHn	Used to represent Unicode CharacterThrough its hex code.

Tokens Escape Sequence Character Literals

Escape Sequence	Functionality	Escape Sequence	Functionality
\’	Used to add a single quote inside a string.	\f	Used to represent formfeed
\”	Used to add double quotes inside a String.	\\	Used to add a backslash
\r	Used for carriage return.	\o	Null Character
\a	Used to add a small beep sound.		

Tokens

String Literals

A string is basically an array of characters. In java, we have a special class for strings that allows users to implement strings to a program very easily. Anything written inside a double quote is a string " ".

String str = "JAVA"; //Valid String Literal

- **Strings in java can also be declared void with a special literal known as null literal. It is basically equivalent to an integer value of 0.**

String null_Literal=null;

Boolean Literals

A boolean literal is a literal that contains only two values true and false. It is declared using the keyword boolean. It is a very useful literal to declare flag variables in different programs.

```
boolean flag1=true;

boolean flag2=false;
```

Tokens

Operators

- **Operators are symbols that perform specific operation on one, two or three operands and then return result**

- Java language supports a rich set of **built-in operators**. An operator is a symbol that tells the compiler to perform a certain mathematical or logical operations, based on the values provided to the operator.
- An **operand** is a value on which any operator works. For example, when we say 7+5, here, numbers 7 and 5 are operands whereas + is an operator

Tokens

Operators

Types of operators	
➢ Arithmetic operators	➢ Unary Operator
➢ Assignment operators	➢ Equality and Relational Operator
➢ Compound Assignment Operator	➢ Conditional operators
➢ Concatenating Operator	➢ Type Comparison Operator
➢ Bitwise Operator	

Arithmetic Operators

Operator	Description	Example (where a and b are variables with some integer value)
+	adds two operands (values)	a+b
-	subtract second operands from first	a-b
*	multiply two operands	a*b
/	divide numerator by the denominator, i.e. divide the operand on the left side with the operand on the right side	a/b
%	This is the **modulus operato**r, it returns the remainder of the division of two operands as the result	a%b

Assignment & Compound Assignment operators

Operator	Description	Example (a and b are two variables, with where a=10 and b=5)
=	assigns values from right side operand to left side operand	a=b, a gets value 5
+=	adds right operand to the left operand and assign the result to left operand	a+=b, is same as a=a+b, value of a becomes 15
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b, is same as a=a-b, value of a becomes 5

Assignment & Compound Assignment operators

Operator	Description	Example (a and b are two variables, with where a=10 and b=5)
*=	mutiply left operand with the right operand and assign the result to left operand	a*=b, is same as a=a*b, value of a becomes 50
/=	divides left operand with the right operand and assign the result to left operand	a/=b, is same as a=a/b, value of a becomes 2
%=	calculate modulus using two operands and assign the result to left operand	a%=b, is same as a=a%b, value of a becomes 0

Concatenating operator

Operator	Description	Example (a and b are two variables String typea)
+	concatenation operator (+) is used to combine two string values to create one new string	a=b+123

Unary Operators

Operator	Description	Example (where a and b are variables with some integer value)
++	This is the **Increment operator** - increases integer value by one. This operator needs only a **single operand**.	a++ or ++a
--	This is the **Decrement operator** - decreases integer value by one. This operator needs only a **single operand**.	--b or b--
-	**Unary minus** converts positive value to negative value	a =-a
+	**Unary plus** is used to represent the **positive** value,that is it makes all bits inverted, every 0 to 1 and every 1 to 0.	a=+a

Equality and Relational Operator

Operator	Description	Example (a and b, where a = 10 and b = 11)
==	Check if two operands are equal	a == b, returns 0
!=	Check if two operands are not equal.	a != b, returns 1 because a is not equal to b
>	Check if the operand on the left is greater than the operand on the right	a > b, returns 0
<	Check operand on the left is smaller than the right operand	a < b, returns 1
>=	check left operand is greater than or equal to the right operand	a >= b, returns 0
<=	Check if the operand on left is smaller than or equal to the right operand	a <= b, returns 1

Conditional /Logical operators

Operator	Description	Example (a and b, where a = 1 and b = 0)
&&	Logical AND	a && b, returns 0
||	Logical OR	a || b, returns 1
!	Logical NOT	!a, returns 0
?:	Ternary	a?1:0

Conditional / Logical operators

- The **ternary operator**, also known as the conditional operators in the java language can be used for statements of the form if-then-else.

(Expression1)? Expression2 : Expression3;

- The question mark ? in the syntax represents the if part.
- The first expression (expression 1) returns either true or false, based on which it is decided whether (expression 2) will be executed or (expression 3)
- If **expression 1** returns true then the **expression 2** is executed.
- If **expression 1** returns false then the **expression 3** is executed.

Type Comparison Operator

- Java **instanceof** operator (also called type comparison operator) is used to test whether the object is an instance of the specified type (class or interface).

Operator	Description	Example
instanceof	Check whether the object is an instance of the specified type	Integer intObj=Integer.*valueOf(v1);* intObj **instanceof Integer**

Bitwise operators

Bitwise operators perform manipulations of data at the bit level.
These operators also perform the shifting of bits from right to left.

Operator	Description	Example
&	Bitwise AND	The bitwise & operator performs a bitwise AND operation
|	Bitwise OR	The bitwise | operator performs a bitwise inclusive OR operation
~	One's complement (NOT)	bitwise NOT operator "~" inverts a bit pattern; it can be applied to any of the integral types, making every "0" a "1" and every "1" a "0"
^	Bitwise Exclusive OR (XOR)	The bitwise ^ operator performs a bitwise exclusive OR operation.
>>	Shift right	int a = 00010000 ;b = 2; a >> b = 00000100
<<	Shift left	int a = 00010000 ;b = 2; a << b = 01000000;

Tokens

Special Symbols

Symbol	Description
Brackets []	These are used as an array element reference and also indicates single and multidimensional subscripts
Parentheses()	These indicate a method call along with method parameters
Braces{}	The opening and ending curly braces indicate the beginning and end of a block of code having more than one statement
Comma (,)	This helps in separating more than one statement in an expression
Semi-Colon (;)	This is used to end any expression or statement

Statements ,Expression & Block

Expressions

An expression is a construct made up of variables, operators, and method invocations, which are constructed according to the syntax of the language, that evaluates to a single value.

```
int cadence = 0;                    anArray[0] = 100;
System.out.println("Element 1 at index 0: " + anArray[0]);
```

Statements

Statements are roughly equivalent to sentences in natural languages. A statement forms a complete unit of execution. The following types of expressions can be made into a statement by terminating the expression with a semicolon (;).

Assignment expressions
Any use of ++ or --
Method invocations
Object creation expressions

Statements ,Expression & Block

A **block** is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed.

```
class BlockDemo {
    public static void main(String[] args) {
        boolean condition = true;
        if (condition) { // begin block 1
            System.out.println("Condition is true.");
        } // end block one
        else { // begin block 2
            System.out.println("Condition is false.");
        } // end block 2
    }
}
```

Java Block

- code defined inside curly brackets { } are called block
- there are 2 types of block
- **1)Static block : Its set of statements which are execute by JVM before the main method**
- **2)Instance block (initializer block): Its Execute whenever object is created ,its execute before constructor call**

Java Block

- A **static block** in a program is a set of statements which are executed by the JVM (Java Virtual Machine) before the main method. At the time of class loading, if we want to perform any task we can define that task inside the static block, this task will be executed at the time of class loading. In a class, any number of a static block can be defined, and this static blocks will be executed from top to bottom.
- In Java, the **initializer block** is used to initialize instance data members. The initializer block is executed whenever an object is created. The Initializer block is copied into Java compiler and then to every constructor. The initialization block is executed before the code in the constructor.

Thanks

Anirudha Gaikwad