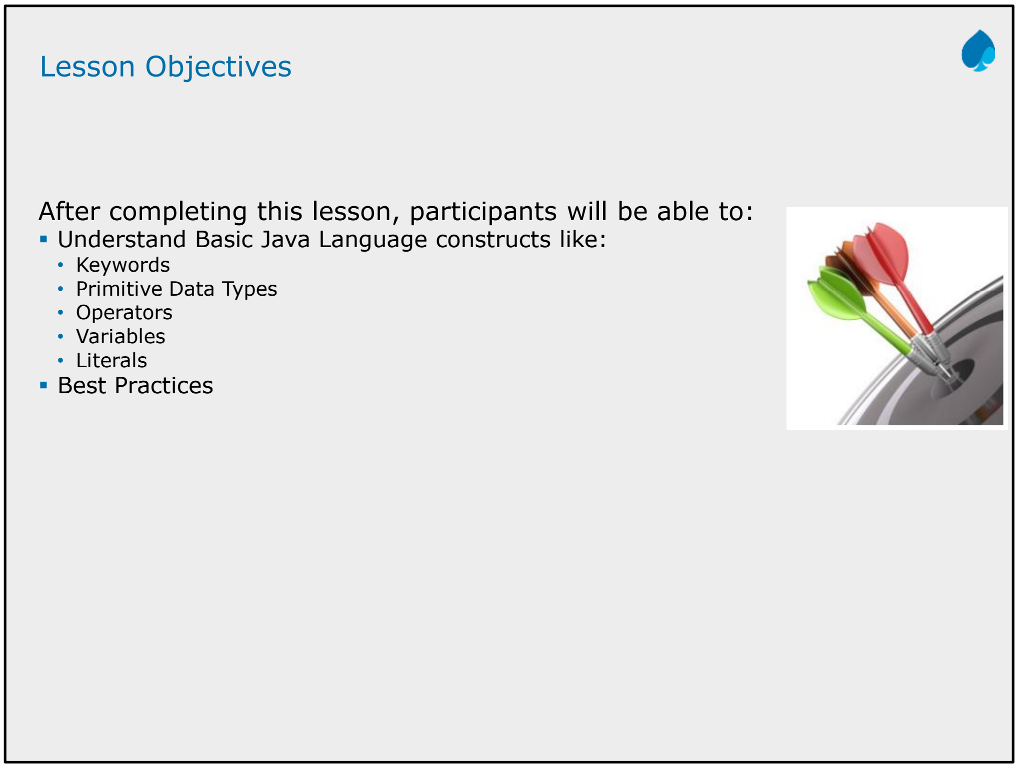
Core Java 8

Lesson 03 : Java Language Fundamentals



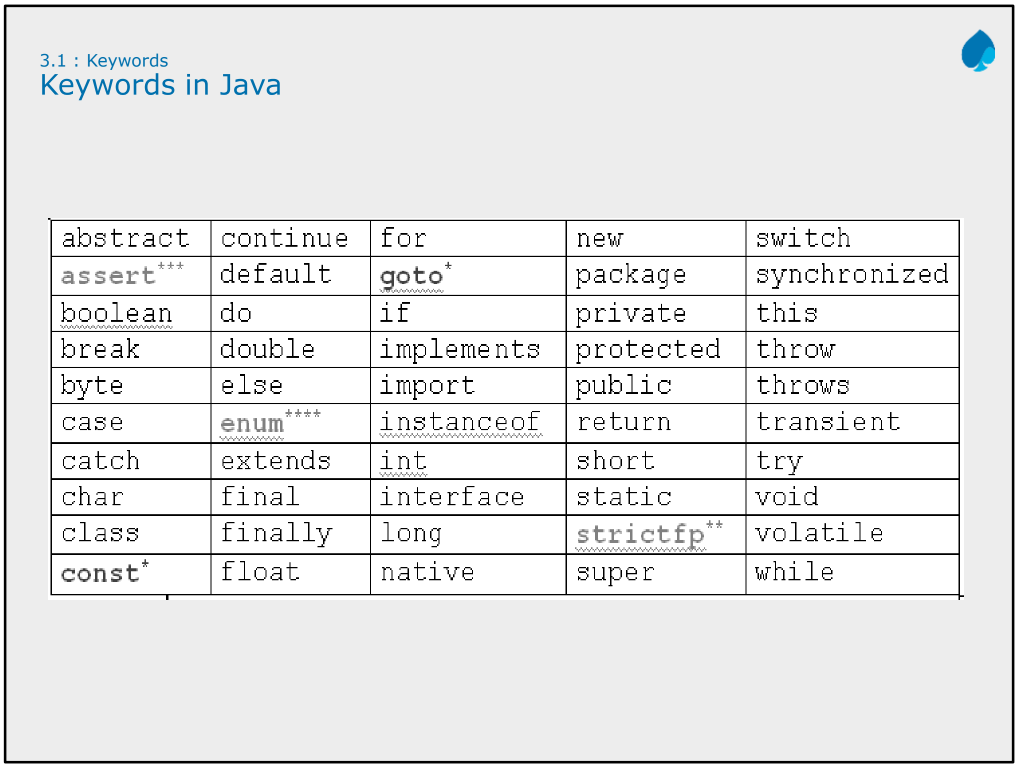
Lesson Objectives

After completing this lesson, participants will be able to:

- Understand Basic Java Language constructs like:
  - Keywords
  - Primitive Data Types
  - Operators
  - Variables
  - Literals
- Best Practices

<u>Lesson Outline:</u>

3.1: Keywords
3.2: Primitive Data Types
3.3: Operators and Assignments
3.4: Variables and Literals
3.5: Flow Control: Java’s Control Statements
3.6: Best Practices



3.1 : Keywords

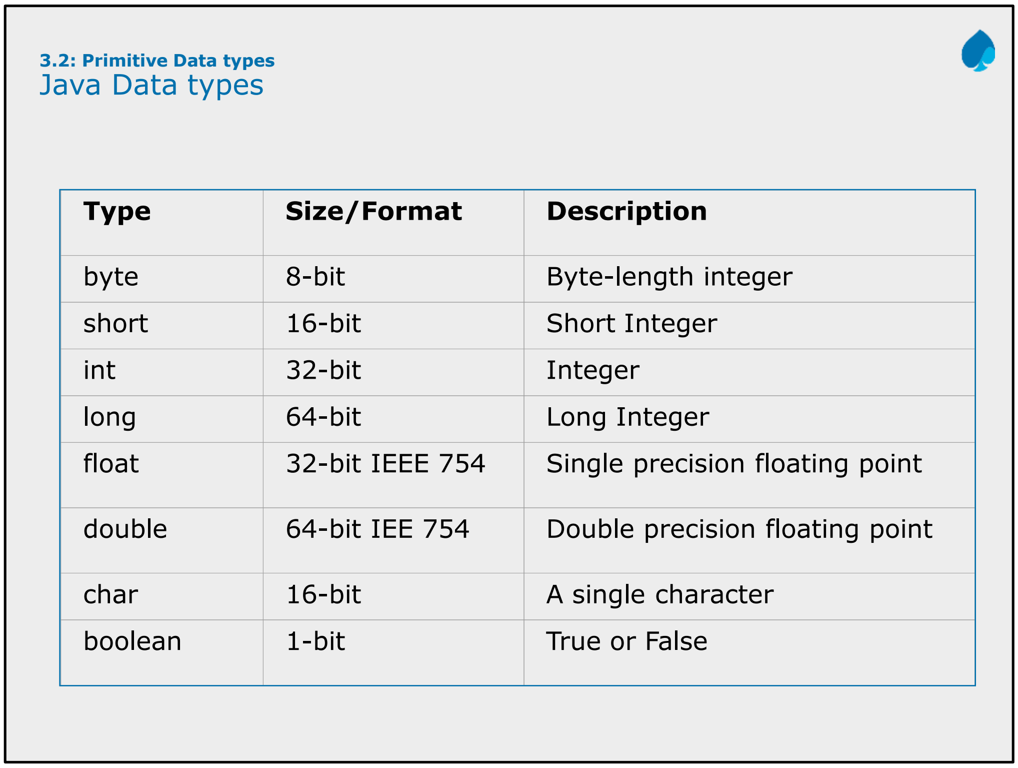
Keywords in Java

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

Keywords are reserved identifiers that are predefined in the language and cannot be used to denote other entities. E.g. class, boolean, abstract, do, try etc. Incorrect usage results in compilation errors.

In addition, three identifiers are reserved as predefined literals in the language: null, true and false.

The table above shows the keywords available in Java 5.



3.2: Primitive Data types

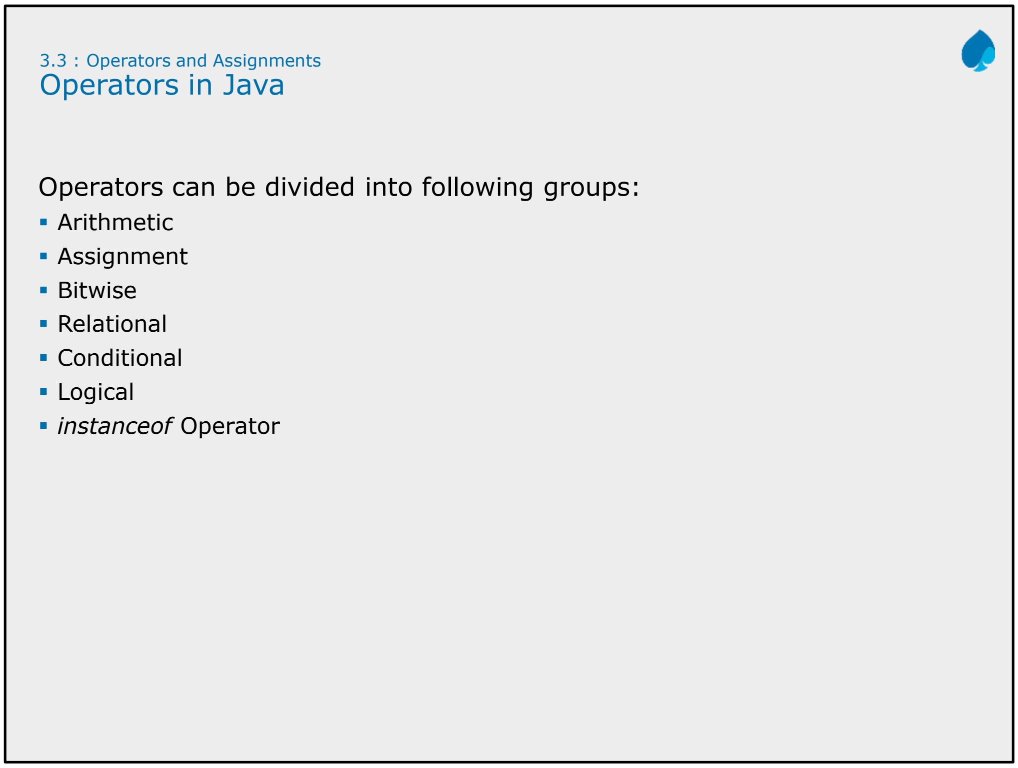
Java Data types

Type	Size/Format	Description
byte	8-bit	Byte-length integer
short	16-bit	Short Integer
int	32-bit	Integer
long	64-bit	Long Integer
float	32-bit IEEE 754	Single precision floating point
double	64-bit IEE 754	Double precision floating point
char	16-bit	A single character
boolean	1-bit	True or False

There are two data types available in Java:
Primitive Data Types.
Reference/Object Data Types :- is discussed later.
There are eight primitive data types supported by Java (see slide above). Primitive data types are predefined by the language and named by a key word.
The default character set used by Java language is **Unicode character** set and hence a **character data type** will consume **two bytes** of memory instead of a byte (a standard for ASCII character set). Unicode is a character coding system designed to support text written in diverse human languages.
This allows you to use characters in your Java programs from various alphabets such as Japanese, Greek, Russian, Hebrew, and so on. This feature **supports** a readymade support for **internalization** of java.
The default values for the various data types are as follows:

Integer	: 0
Character	: '\u0000'
Decimal	: 0.0
Boolean	: false
Object Reference:	null

[Note: C or C++ data types pointer, struct, and union are not supported. Java does not have a typedef statement (as in C and C++).]



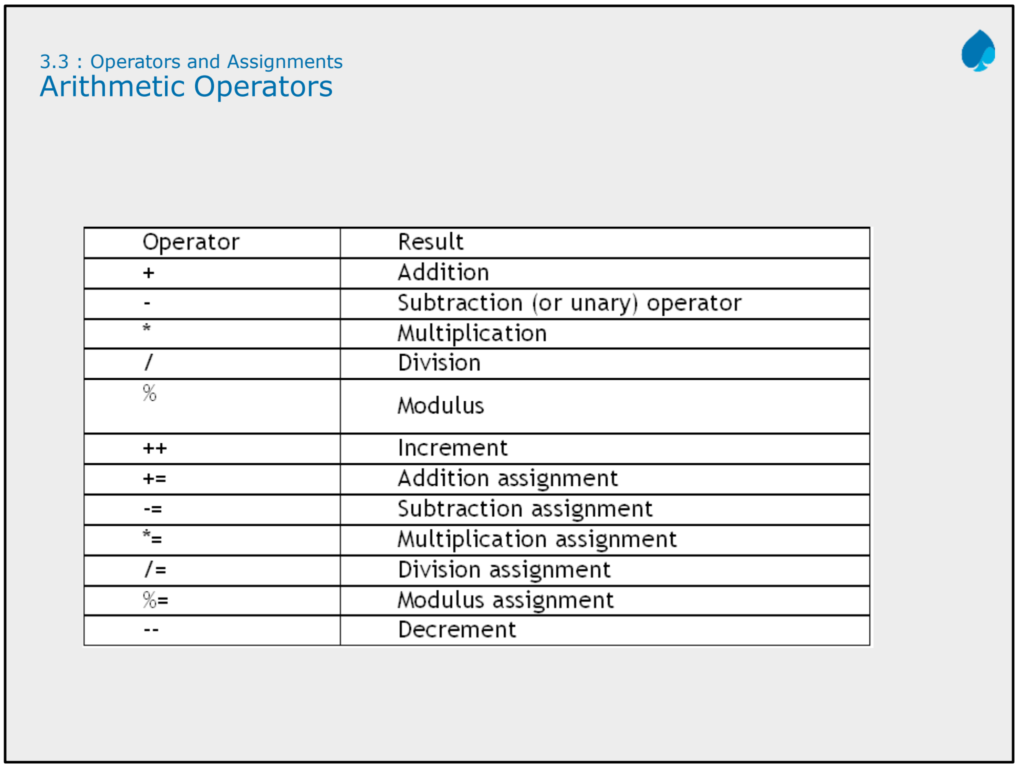
3.3 : Operators and Assignments

Operators in Java

Operators can be divided into following groups:

- Arithmetic
- Assignment
- Bitwise
- Relational
- Conditional
- Logical
- *instanceof* Operator

Java provides a rich set of operators to manipulate variables. These are classified into several groups as shown above.



3.3 : Operators and Assignments

Arithmetic Operators

Operator	Result
+	Addition
-	Subtraction (or unary) operator
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
-=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
--	Decrement

Arithmetic operators are summarized in the table above:
Integer division yields an integer quotient for example, the expression 7 / 4 evaluates to 1, and the expression 17 / 5 evaluates to 7. Any fractional part in integer division is simply discarded (i.e., truncated) no rounding occurs.
Java provides the remainder operator, %, which yields the remainder after division. The expression x % y yields the remainder after x is divided by y. Thus, 7 % 4 yields 3, and 17 % 5 yields 2. This operator is most commonly used with integer operands, but can also be used with other arithmetic types.
Parentheses are used to group terms in Java expressions in the same manner as in algebraic expressions. For example, to multiply a times the quantity b + c, we write **a*(b+c).**
If an expression contains nested parentheses, such as **((a+b)*c)** the expression in the innermost set of parentheses (a + b in this case) is evaluated first.
Order of Precedence:
Multiplication, division and remainder operations are applied first. If an expression contains several such operations, the operators are applied from left to right.
Multiplication, division and remainder operators have the same level of precedence.
Addition and subtraction operations are applied next. If an expression contains several such operations, the operators are applied from **left to right**. **Addition and subtraction operators have the same level of precedence.**

Assignment Operators

Assignment operator is used to assign value to a variable.

This is the operator you are already familiar with. This is denoted by the symbol “=”. This is used to assign the value to a variable.

Example

int value = 10;

Shorthand Assignment Operators :

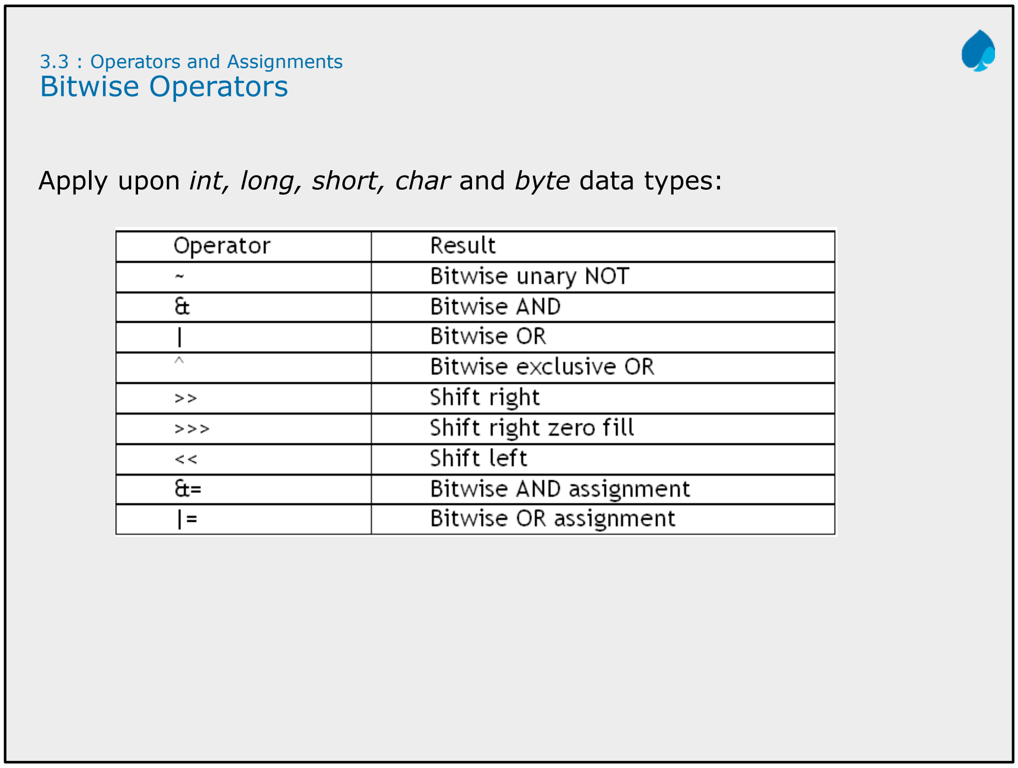
+= operator : var1=var1+var2 → var1+=var2;

-= operator : var1= var1-var2 → var1-+var2;

*= operator : var1=var1*var2 → var1*=var2;

/= operator : var1=var1/var2 → var1/=var2;

%= operator : var1= var%var2 → var1%=var2;



3.3 : Operators and Assignments

Bitwise Operators

Apply upon *int, long, short, char* and *byte* data types:

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment

The Java programming language also provides operators that perform bitwise and bit shift operations on integral types. The operators discussed in this section are less commonly used.

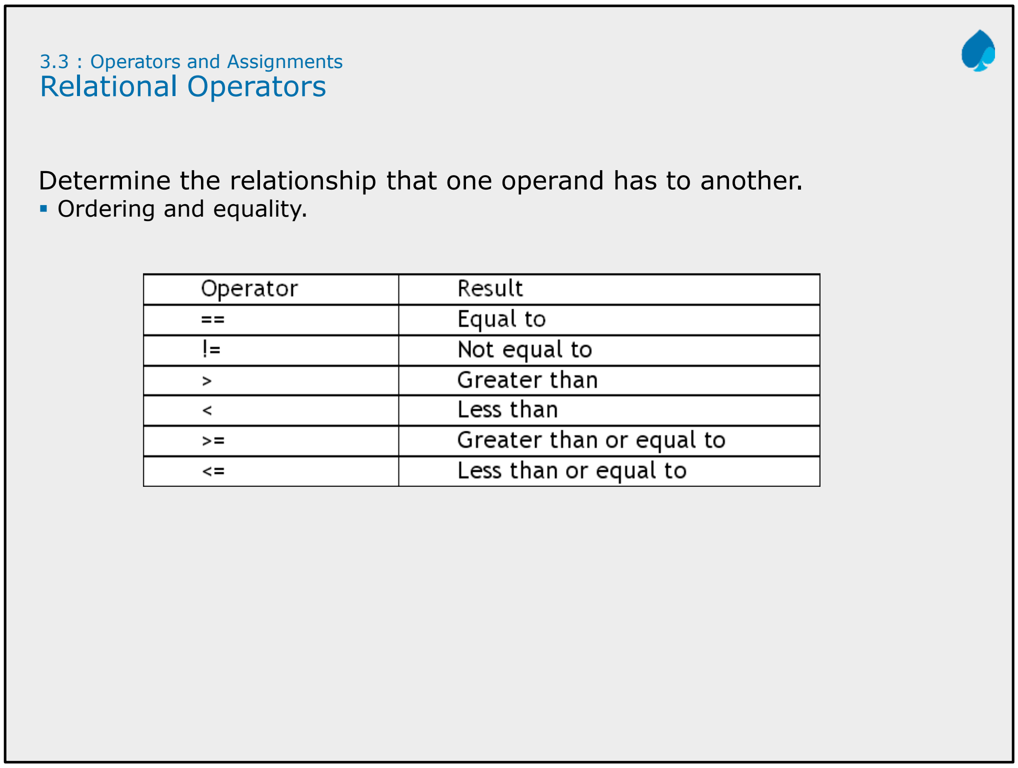
The unary bitwise complement operator "~" inverts a bit pattern; it can be applied to any of the integral types, making every "0" a "1" and every "1" a "0". For example, a byte contains 8 bits; applying this operator to a value whose bit pattern is "00000000" would change its pattern to "11111111".

The signed left shift operator "<<" shifts a bit pattern to the left, and the signed right shift operator ">>" shifts a bit pattern to the right. The bit pattern is given by the left-hand operand, and the number of positions to shift by the right-hand operand. The unsigned right shift operator ">>>" shifts a zero into the leftmost position, while the leftmost position after ">>" depends on sign extension.

Bitwise & operator performs a bitwise AND operation.

Bitwise ^ operator performs a bitwise exclusive OR operation.

Bitwise | operator performs a bitwise inclusive OR operation.



3.3 : Operators and Assignments

Relational Operators

Determine the relationship that one operand has to another.

- Ordering and equality.

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

A condition is an expression that can be either true or false. For example, the condition "grade is greater than or equal to 60" determines whether a student passed a test. If the condition in an if statement is true, the body of the if statement executes. If the condition is false, the body does not execute.

Conditions in if statements can be formed by using the equality operators (== and !=) and relational operators (>, <, >= and <=). Both equality operators have the same level of precedence, which is lower than that of the relational operators. The equality operators associate from left to right. The relational operators all have the same level of precedence and also associate from left to right.

Conditional Operator

This operator is used to make conditional expressions.

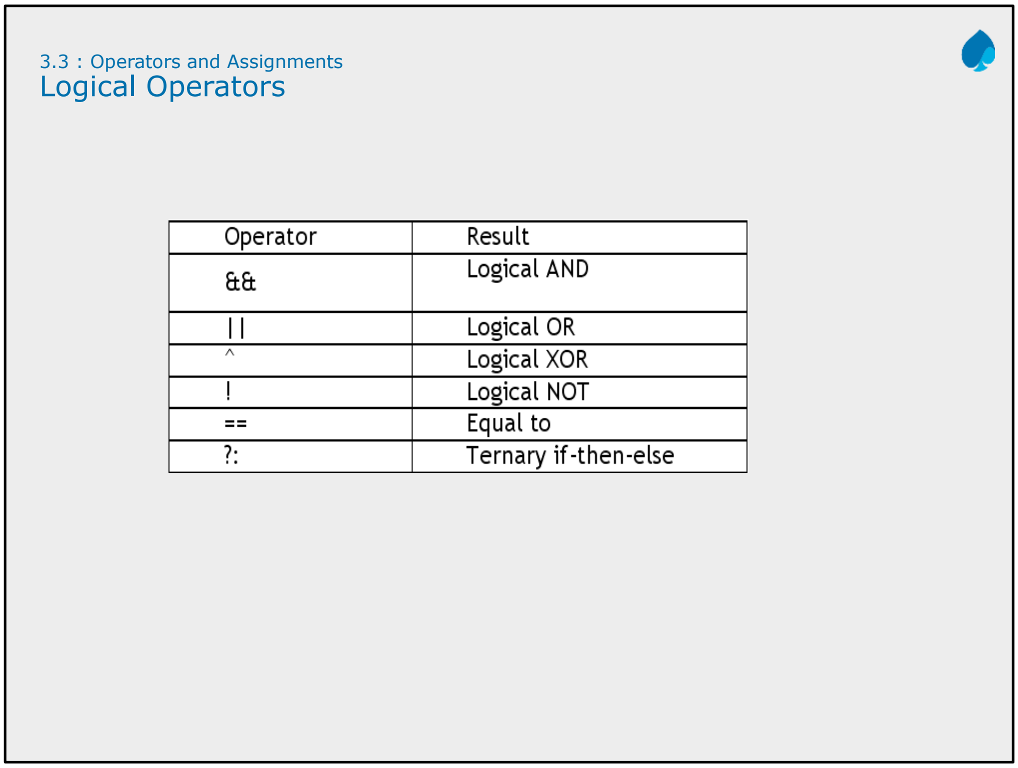
Syntax :

Expression1 ? Expression 2 : expression 3 ;

Here expression1 will be evaluated first. IF we get "true" then the result of expression2 will be overall result of conditional expression .

If we get "false" then expression3 will be evaluated and the result of expression3 will be overall result of conditional expression .

```
int a=10;
int b= 20;
int value1 = (a<b) ? a : b;
int value2 = (a>b) ? a : b;
```

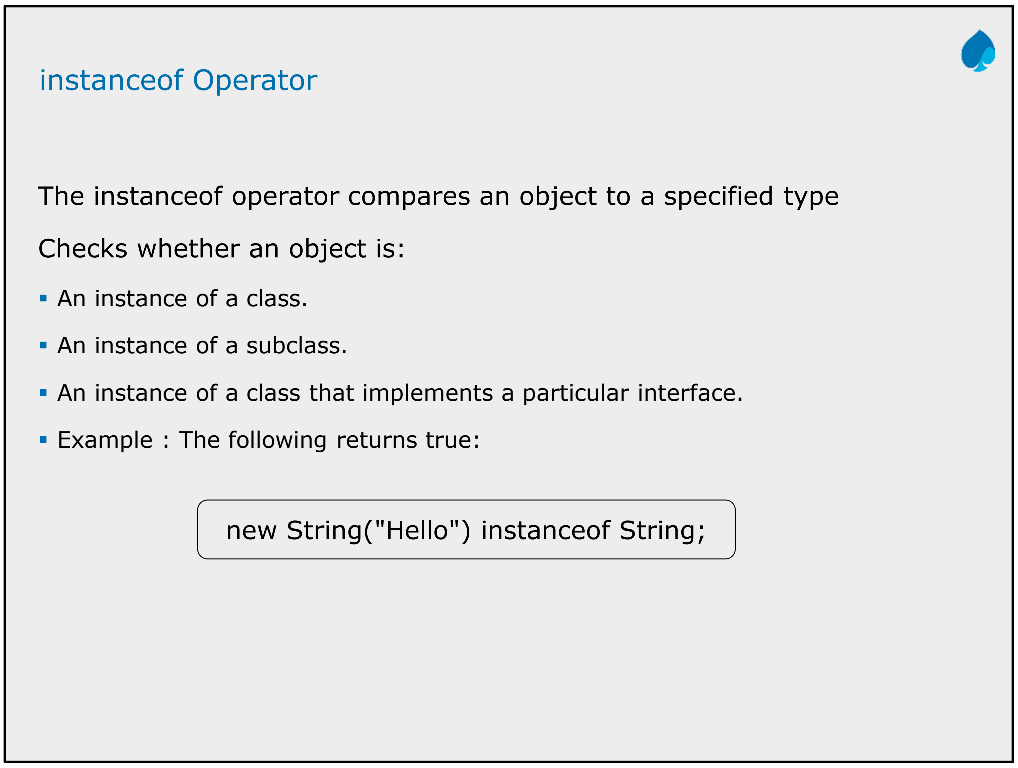


3.3 : Operators and Assignments

Logical Operators

Operator	Result
&&	Logical AND
||	Logical OR
^	Logical XOR
!	Logical NOT
==	Equal to
?:	Ternary if-then-else

Java provides logical operators to enable programmers to form more complex conditions by combining simple conditions. The logical operators are && (conditional AND), || (conditional OR), & (boolean logical AND), | (boolean logical inclusive OR), ^ (boolean logical exclusive OR) and ! (logical NOT).



instanceof Operator

The instanceof operator compares an object to a specified type

Checks whether an object is:

- An instance of a class.
- An instance of a subclass.
- An instance of a class that implements a particular interface.
- Example : The following returns true:

```
new String("Hello") instanceof String;
```

The instanceof operator is used to make a test whether the given object belongs to specified type. Consider the below example. The if statement returns true here as the child object is type of its superclass.

```
class Ticket{
}
class ConfirmedTicket extends Ticket {
}
…
…
ConfirmedTicket tkt= new ConfirmedTicket();
If(tkt instanceof Ticket) {
    //some processing
}
```

3.4: Variables and Literals

Variables

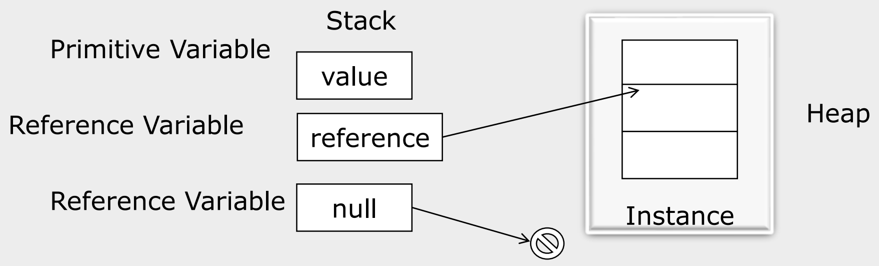
Variables are data placeholders.

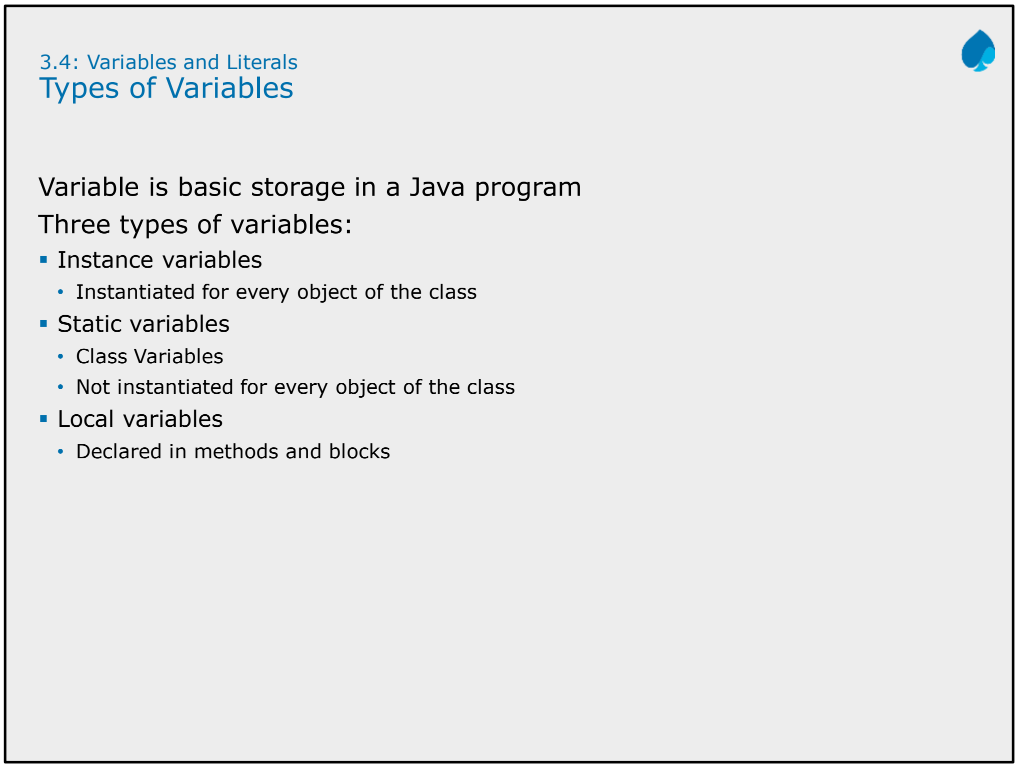
Java is a strongly typed language, therefore every variable must have a declared type.

The variables can be of two types:

- reference types: A variable of reference type provides a reference to an object.
- primitive types: A variable of primitive type holds a primitive.

In addition to the data type, a Java variable also has a name or an identifier.





3.4: Variables and Literals

Types of Variables

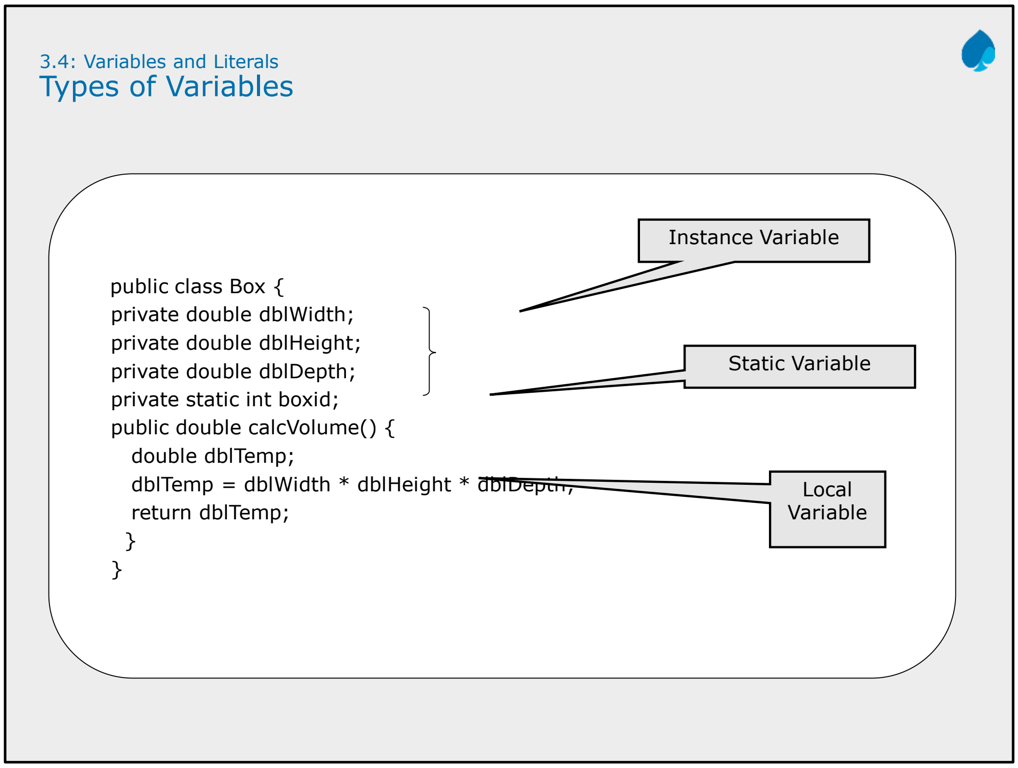
Variable is basic storage in a Java program

Three types of variables:

- Instance variables
  - Instantiated for every object of the class
- Static variables
  - Class Variables
  - Not instantiated for every object of the class
- Local variables
  - Declared in methods and blocks

Instance variables: These are members of a class and are instantiated for every object of the class. The values of these variables at any instant constitute the *state* of the object.
Static variables: These are also members of a class, but these are not instantiated for any object of the class and therefore belong only to the class. We shall be covering the static modifier in later section.
Local variables: These are declared in methods and in blocks. They are instantiated for every invocation of the method or block. In Java, local variables must be declared before they are used.
Life-cycle of the variable is controlled by the scope in which those are defined.

Refer the example on the subsequent slide.



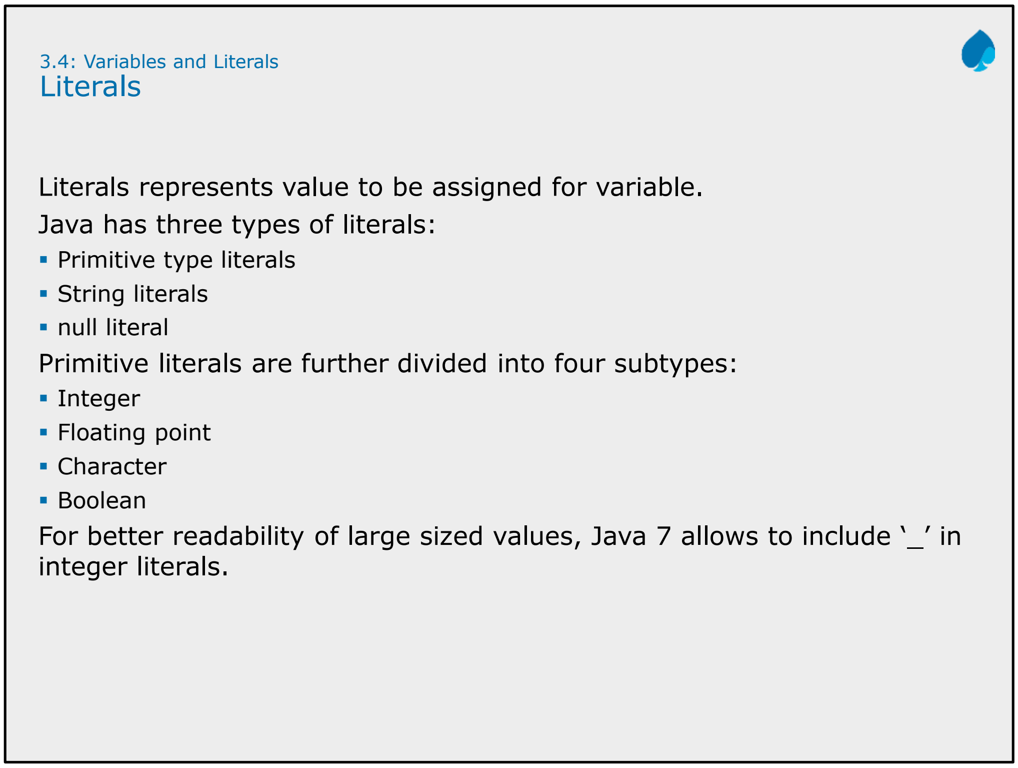
3.4: Variables and Literals

Types of Variables

```
public class Box {
private double dblWidth;
private double dblHeight;
private double dblDepth;
private static int boxid;
public double calcVolume() {
   double dblTemp;
   dblTemp = dblWidth * dblHeight * dblDepth;
   return dblTemp;
  }
}
```

Instance Variable
Static Variable
Local Variable

Add the notes here.



3.4: Variables and Literals

Literals

Literals represents value to be assigned for variable.

Java has three types of literals:

- Primitive type literals
- String literals
- null literal

Primitive literals are further divided into four subtypes:

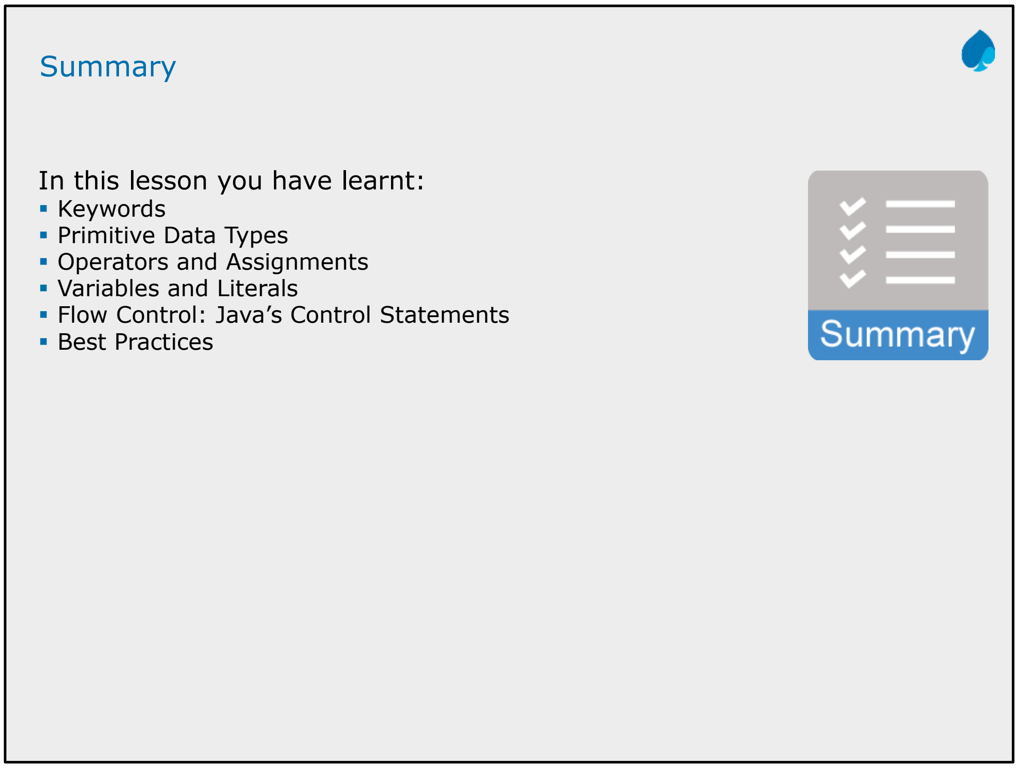
- Integer
- Floating point
- Character
- Boolean

For better readability of large sized values, Java 7 allows to include '_' in integer literals.

Literal Type	Example
Integer	int x = 10
Octal	int x = 0567
Hexadecimal	int x = 0x9E (to represent number 9E)
Long	long x = 9978547210L;
Binary	byte twelve = 0B1100; (to represent decimal 12)

Using Underscores	int million = 1_000_000;
	int twelve = 0B_1100;
	long multiplier = 12_34_56_78_90_00L;

Float	float x = 0.4f; float y = 1.23F, float z = 0.5e10;
Double	double x = 0.0D; double pi=3.14; double z=9e-9d;
Boolean	boolean member=true; boolean applied=false;
Character	char gender = ‘m’;
String	String str = “Hello World”;
Null	Employee emp = null;



Summary

In this lesson you have learnt:

- Keywords
- Primitive Data Types
- Operators and Assignments
- Variables and Literals
- Flow Control: Java’s Control Statements
- Best Practices

Summary

Review Question

Question 1: Java considers variable number and NuMbEr to be identical.

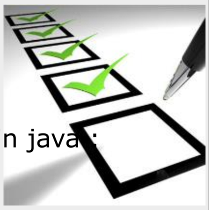
- True/False

Question 2: The *do...while* statement tests the loop-continuation condition ___________ it executes executing the loop's body; hence, the body executes at least once.

- **Option1:** before
- **Option2:** after

Review Question

Question 1: What is the output of below expression.
6-2+10%4+7
a.10
b.12
c.13
d.14



Question 2: Which of the Following Operator does not exist in java:
1. >>
2. %=
3. >>>
4. <<