Object Oriented Programming using Java

Prepared By:
Suyel, PhD
Assistant Professor
Dept. of CSE, NIT Patna

Outline

1. Data Type
2. Tokens

Data Type

- ❑ Data type specifies the size and type of values that can be stored in an identifier.
- ❑ Java language is rich in its data types.
- ❑ Different data types allow us to select the appropriate type.
- ❑ There are mainly two types of data type:
  1. **Primitive:** Byte, short, int, long, float, double, boolean and char.
  2. **Non-primitive:** String, arrays, class and interface.

Data Type (Cont…)

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Data Type (Cont…)

- The range of values is calculated as $-(2^{n-1})$ to $(2^{n-1})-1$, where n is the number of bits required.
- Example:

  The byte data type requires 1 byte = 8 bits. Therefore, the range of values that can be stored in the byte data type is:

  $-(2^{8-1})$ to $(2^{8-1})-1$

  $= -2^7$ to $(2^7) -1$

  $= -128$ to 127

Data Type (Cont…)

❑ Differences between primitive and non-primitive data types:

Primitive	Non-primitive
Primitive types are predefined (already defined) in java.	Non-primitive types are created by the programmer (except for String).
Primitive types cannot be used to call methods to perform certain operations.	Non-primitive types can be used to call methods to perform certain operation.
A primitive type has always a value.	Non-primitive types can be null.

Tokens

- A **token** is the smallest element of a program that is meaningful to the compiler.
- Java compiler breaks the line of code into text (words), which is called java tokens.
- The java compiler identifies these words as tokens.
- These tokens are separated by the delimiters. It is useful for compilers to detect errors.
- Java token includes Keywords, Identifiers, Literals, Operators and Separators.

Tokens (Cont…)

❑ **Keywords**

- These are the pre-defined reserved words of any programming language.
- **Keywords** have special meaning to the java compiler.
- Each keyword is assigned a special task or function and cannot be changed by the user.
- Since keywords are referred names for the compiler, they can't be used as variables or identifiers.
- It is always written in lower case.

Tokens (Cont…)

❑ Keywords (Cont…)

01. abstract	02. boolean	03. byte	04. break	05. class
06. case	07. catch	08. char	09. continue	10. default
11. do	12. double	13. else	14. extends	15. final
16. finally	17. float	18. for	19. if	20. implements
21. import	22. instanceof	23. int	24. interface	25. long
26. native	27. new	28. package	29. private	30. protected
31. public	32. return	33. short	34. static	35. super
36. switch	37. synchronized	38. this	39. throw	40. throws
41. transient	42. try	43. void	44. volatile	45. while
46. assert	47. const	48. enum	49. goto	50. strictfp

Tokens (Cont…)

❑ Identifier

- ❖ Java **Identifiers** are the user-defined names of variables, methods, classes, arrays, packages and interfaces.
- ❖ Once we assign an identifier in a java program, we can use it to refer the value associated with that identifier in later statements.
- ❖ There are some rules for naming the identifiers:
  1. The first letter of an identifier must be a letter, underscore or dollar sign. It cannot start with digits, but may contain digits.
  2. The only special characters that are allowed are “$” and “_”.
  3. Identifier name cannot contain white spaces.
  4. Identifiers are case sensitive.
  5. Java identifiers can be of any length.
  6. Most importantly, keywords cannot be used as identifiers in java.

Tokens (Cont…)

- **Identifier (Cont…)**
  - Examples:

x1	//Valid
i1	//Valid
_myvariable	//Valid
$myvariable	//Valid
sum_of_array	//Valid
hi123	//Valid
My Variable	//Contains a space
123hello	//Begins with a digit
a+c	//Plus sign is not an alphanumeric character

Tokens (Cont…)

Literals

- **Literals** in java are similar to normal variables, but their values cannot be changed once assigned.
- In other words, literals are constant variables with fixed values.
- These are defined by users and can belong to any data type.
- Java supports five types of literals that are as follows:
  1. Integer
  2. Floating point
  3. Character
  4. String
  5. Boolean

Tokens (Cont…)

❑ Literals (Cont…)

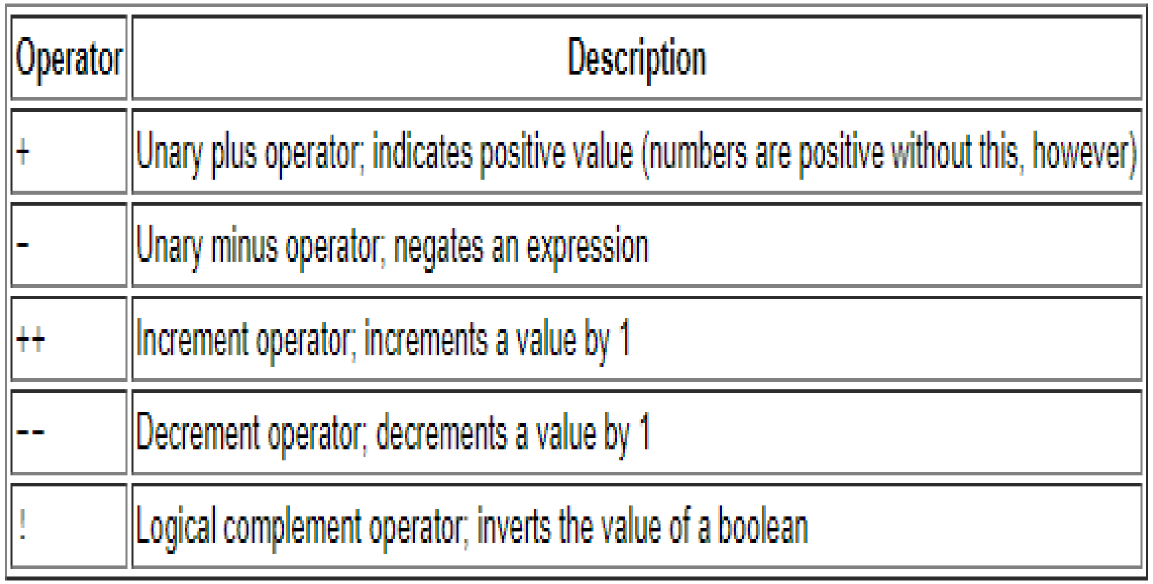
Literal	Type
23	int
9.86	double
false, true	boolean
'K', '7', '-'	char
"javatpoint"	String

Tokens (Cont…)

- **Operators**
  - **Operators** are the special symbol that tells the compiler to perform a special operation.
  - Java provides different types of operators that can be classified according to the functionality they provide.
  - There are many operators in java:
    1. Unary operators
    2. Arithmetic operators
    3. Relational operators
    4. Bitwise operators
    5. Logical operators
    6. Assignment operators
    7. Ternary operators

Tokens (Cont…)

- **Operators (Cont…)**
  - Unary operators



Operator	Description
+	Unary plus operator; indicates positive value (numbers are positive without this, however)
-	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean

Tokens (Cont…)

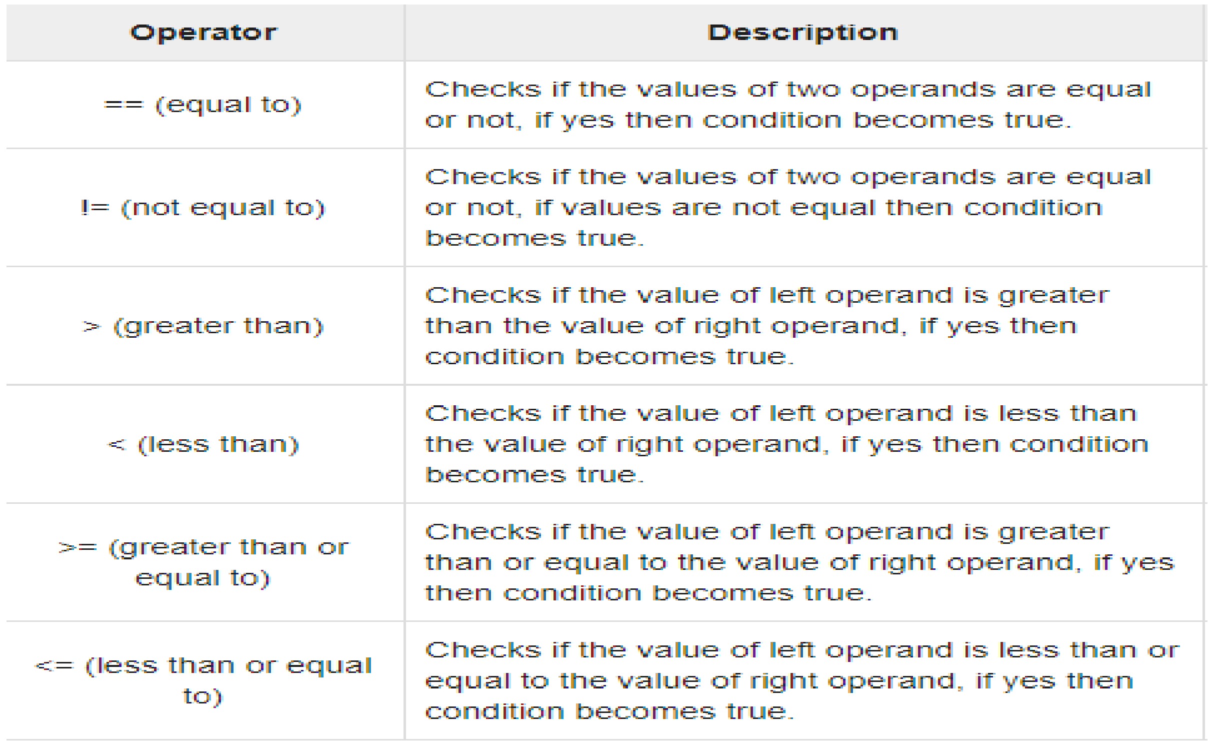
- **Operators (Cont…)**
  - Arithmetic operators

Operator	Description
+ (Addition)	Adds values on either side of the operator.
- (Subtraction)	Subtracts right-hand operand from left-hand operand.
* (Multiplication)	Multiplies values on either side of the operator.
/ (Division)	Divides left-hand operand by right-hand operand.
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.

Tokens (Cont…)

❑ Operators (Cont…)

❖Relational operators



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

Tokens (Cont…)

❑ Operators (Cont…)

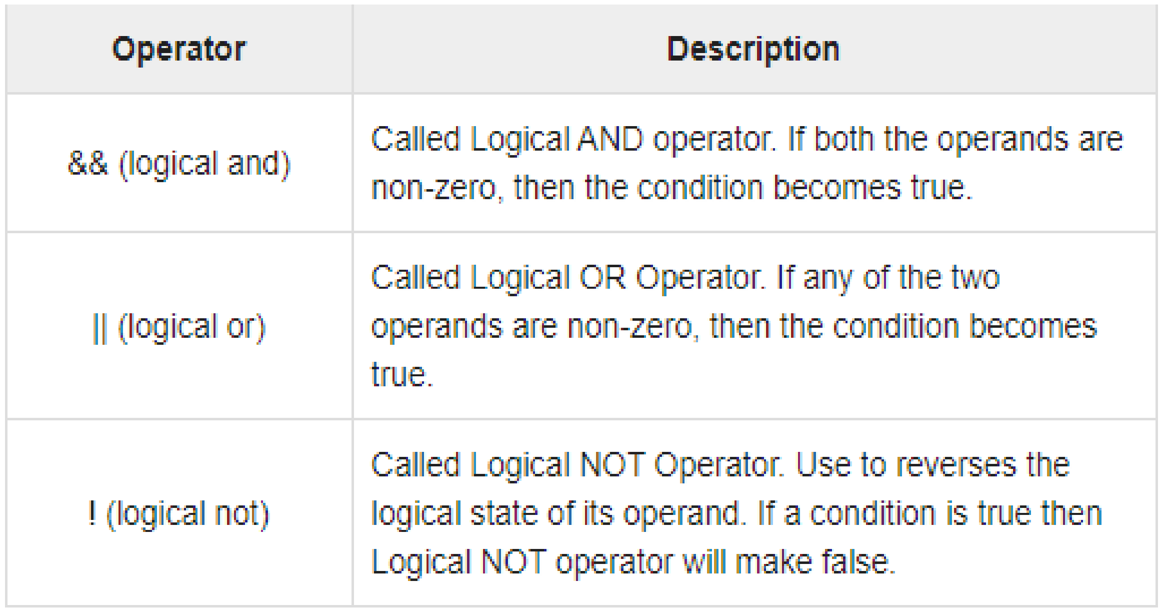
❖ Bitwise operators

Operator	Description
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.
>>> (zero fill right shift)	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.

Tokens (Cont…)

❑ **Operators (Cont…)**

❖ Logical operators



Operator	Description
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.
! (logical not)	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.

Tokens (Cont…)

- **Operators (Cont…)**
  - Assignment operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Tokens (Cont…)

- **Operators (Cont…)**
  - Ternary operators
    - This operator is a shorthand version of if-else statement and also known as **Conditional Operator**.
    - It has three operands. So, the name isternary. General form:

variable = Expression ? expression1 : expression 2

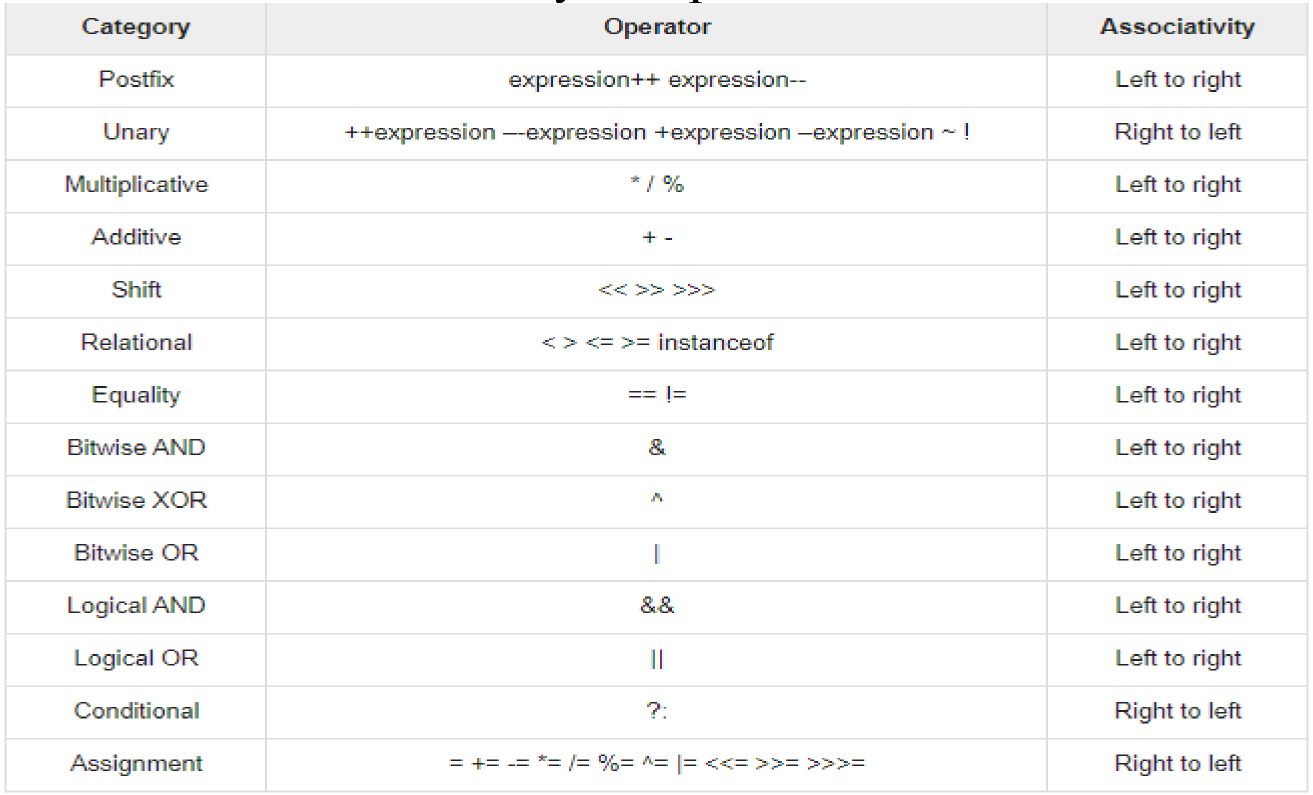
If the Expression is true, expression1 is assigned to the variable.

If the Expression is false, expression2 is assigned to the variable.

Tokens (Cont…)

❑ Operators (Cont…)

- Precedence and Associativity of Operators



Category	Operator	Associativity
Postfix	expression++ expression--	Left to right
Unary	++expression —-expression +expression –expression ~ !	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >> >>>	Left to right
Relational	< > <= >= instanceof	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= ^= |= <<= >>= >>>=	Right to left

Tokens (Cont…)

- **Separators**
  - **Separators** are symbols that indicate the division and arrangement of groups of code. The structure and function of code is generally defined by the separators. The separators used in java are as follows:
    - **Parentheses “( )”:** It is used to define precedence in expressions to enclose parameters in method definitions and enclosing cast types.
    - **Braces “{ }”:** Braces are used to define a block of code and to hold the values of arrays.
    - **Brackets “[ ]”:** These are used to declare array types.
    - **Semicolon “;”:** Semicolon is used to separate statements.
    - **Comma “,”:** It is used to separate identifiers in a variable declaration and in the for loop.
    - **Period “.”:** Period is used to separate package names from classes and subclasses and to separate a variable or a method from a reference variable.

thank
you . . .

Slides are prepared from various sources, such as Book, Internet Links and many more.