CHAPTER 4 PROGRAMMING FUNDAMENTALS

TOKENS

The smallest individual unit a program is known as a **Token**. Java has following types of tokens:

- Keywords
- Identifiers
- Literals
- Punctuators/Separators
- Operators

KEYWORDS

- These are those words that convey a special meaning to language compiler. These are reserved for special purpose and **must not be used as a identifier name.**
- e.g. int , float, double, char, String , break are some examples of keyword.

IDENTIFIERS

- These are the fundamental building block of a program and are used as the general terminology for the names given to different parts of the program namely, variables etc.

Identifiers forming rules of java:

1. Identifiers can have alphabets, digits and underscore and dollar sign characters.

2. They must not be a keyword.

3. They must not begin with a digit.

4. They can be of any length.

5. Java is case-sensitive i.e. upper-case and lower-case letters are treated differently.

Examples of some valid identifiers are:

myfile	$abc
abc_12	_abc

LITERALS

Also referred as constants, and are data items that are fixed data values. Literals available in Java are:

- Integer-literal
- Floating-literal
- Boolean-literal
- Character-literal
- String-literal
- Null-literal

Integer-Literal

- These are the whole numbers without any fractional part. **Rules of writing integer constants are:**

1. It must have at least one digit and must not contain any decimal points.
2. Commas cannot appear in an integer constant.

3. It may contain either + or – sign.

- Examples of valid integer literals are: 56 , +8902 , -235 , 090 etc.

- Java allows **three types of integer literals:**

    1. Decimal (base 10)

    2. Octal (base 8)

    3. HexaDecimal (base 16)

Floating-Literals (Real Literals)

- Real literals are having fractional parts. **Rules of writing floating literals are :**

1. It must have at least one digit before a decimal point and at least one digit after the decimal point.

2. Commas cannot appear in a Floating literal.

3. It may contain either + or – sign.

- **Examples of valid floating literals are**: 2.0 , +17.5 , -13.0 , -0.000875 etc.

- Real literals may be written in two forms: **FRACTIONAL FORM , EXPONENT FORM**

REAL LITERAL IN EXPONENT FORM

A real literal in exponent form consists of two parts: **mantissa and exponent.**
e.g.

5.8 can be written in exponent form as: $0.58 \times 10^1 = 0.58E1$

Where mantissa part is 0.58(the part appearing before E) and exponent part is 1 (the part appearing after E).

Boolean-literal

- A boolean literal is having two values only i.e. **true and false.**

Character-literal

- It is one character enclosed in single quotes, e.g. 'z' . **Rule of writing character literals are :**

1. It must contain one character and must be enclosed in single quotation marks.

e.g. 'C' , 'y' , '1' , '8' , '#' etc. \60 9

NONGRAPHIC CHARACTERS

- Those characters that cannot be typed directly from keyboard e.g. backspace, tabs etc.

- These characters are represented by using escape sequence. **An escape sequence is represented by a backslash followed by one or more characters.**

E.g.

\n – represents next line escape sequence.
\t – represents horizontal tab escape sequence.
\v – represents vertical tab escape sequence.

String-literals

- Multiple character constants are treated as string literal. **Rule of forming String-Literals are :**
    1. It is a sequence of zero or more characters surrounded by double quotes.
    e.g. “abc” , “raj” , “ko123” , “123” etc.

Null-literals

- A null literal is having value ***null*** only.

SEPARATORS

- The following **nine characters** are the separators :

( ) { } [ ] ; , .

OPERATORS

- Operator are those character that specify a particular operation.
- E.g. **+ , - , * , < , >** etc.
- Java has total **37 operators.**

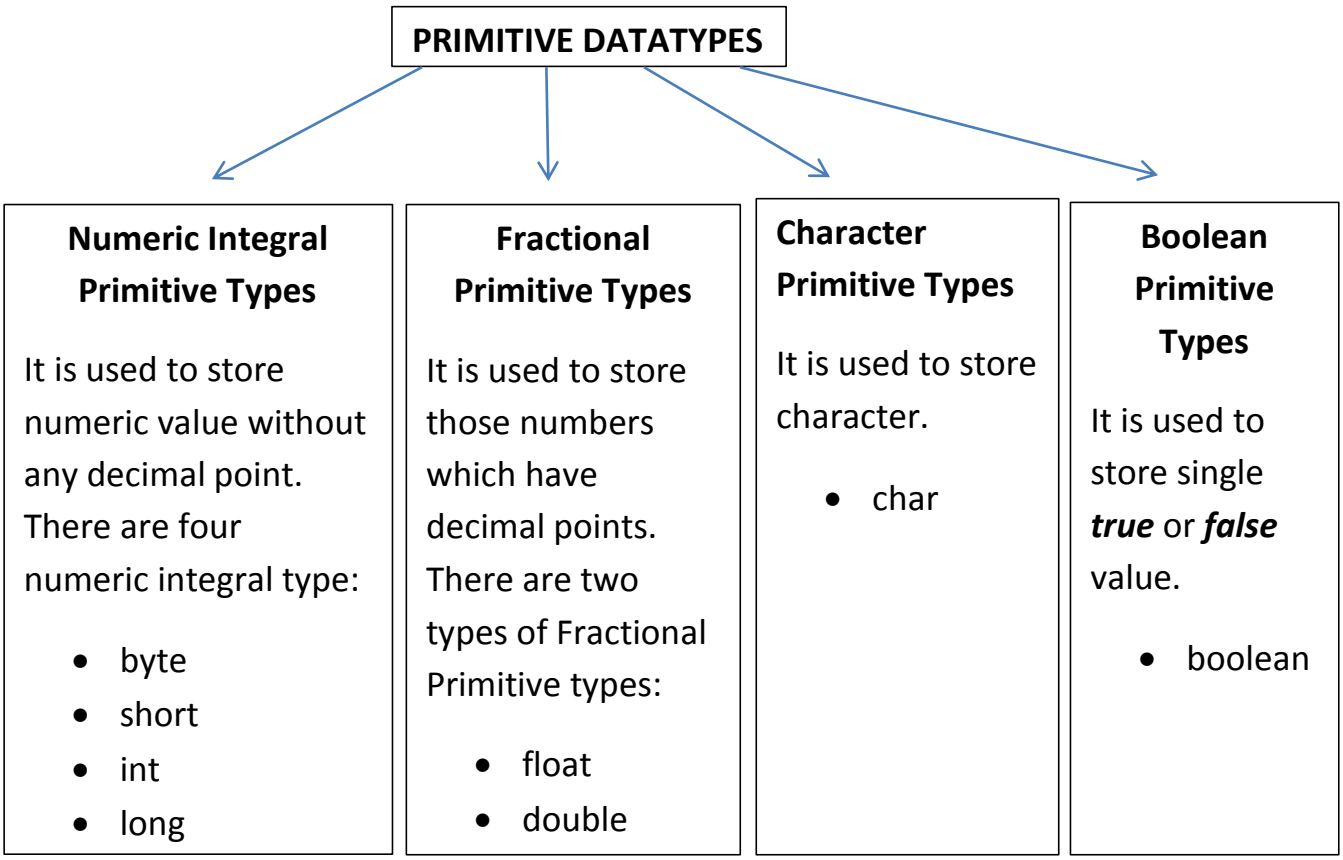
DATA TYPES

- Data can be many types e.g. character, integer, real, string etc., and in order to handle different type of data, java provides Data types.
- Data types are means to identify the type of data and associated operations of handling it.

- Java data types are of **two types :** **PRIMITIVE DATA TYPE** , **REFERENCE DATA TYPES**

PRIMITIVE DATA TYPES (FUNDAMENTAL DATA TYPES)

- It comes as part of the language. Java provides eight primitive datatypes which are : byte, short , int, long,float, double, char, Boolean.



SIZE & RANGE OF EACH DATATYPE

TYPE	SIZE	RANGE
byte	8 bits (1 byte)	-128 to +127
short	16 bits (2 bytes)	-32768 to +32767
int	32 bits (4 bytes)	-2^{31} to $2^{31}-1$
long	64 bits (8 bytes)	-2^{63} to $2^{63}-1$
float	32 bits (4 bytes)	-3.4E+38 to +3.4E+38

double	64 bits (8 bytes)	-1.7E+308 to +1.7E+308
char	16 bits (2 bytes)	0 to 65536
boolean	1 bit	true or false

SOME DATA VALUES AND THEIR DATA TYPES

VALUE	DATATYPE
178	
8864	
37.26	
87.636	
'c'	
true	
false	

REFERENCE DATA TYPES

- These are constructed from primitive data types. **E.g. classes, array and interface**.

VARIABLES

- A variable is a named memory location, which holds a data value of a particular data type.

  e.g. the following statement declares a variable I of the data type int :

int i ;

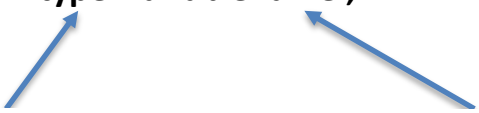
Declare a variable **j** of **float** data type:

Declare a variable **ch** of **char** data type:

DECLARATION OF VARIABLES

- The declaration of a variable generally takes the following form:

type variablename ;



Any Valid Java Data type

It is the name of variable. A *variablename* is an identifier. Thus all rules of identifier naming apply to the name of a variable.

- E.g. Following declaration creates a variable ***age*** of ***int*** type :

int age ;

- When **more than one variable of same data type** needs to declared then we can write the declaration as :

int year, day , month ;

double salary , wage ;

INITIALISATION OF VARIABLES

- All examples given above in declaration of variable, does not provide initial value or first value to variable.
- A variable with declared first value is said to be an initialised variable. E.g.

int age = 18 ;

double price = 2500.35 ;

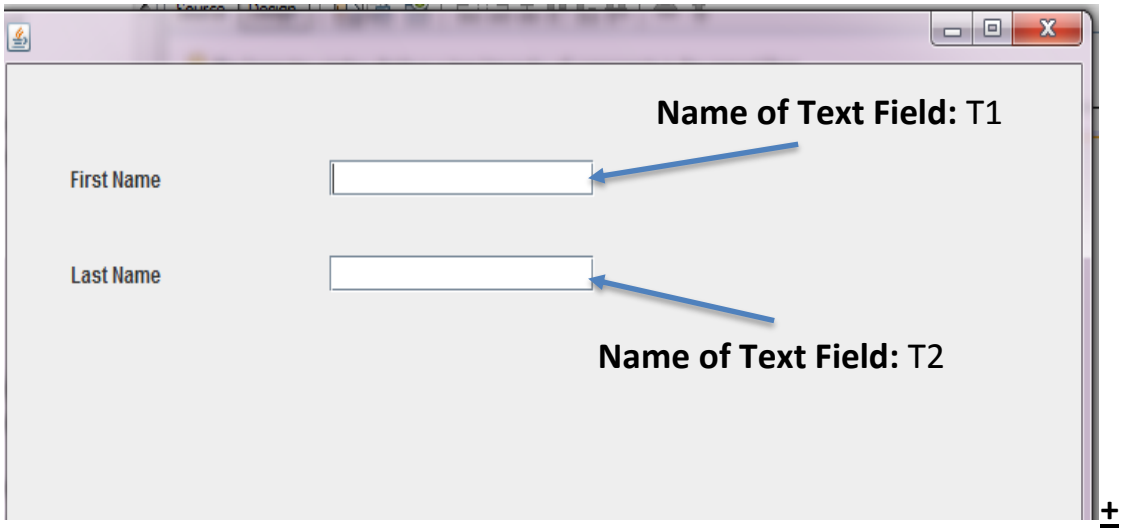
TEXT INTERACTION IN GUIs

- For text interaction in a GUI, **four types of methods are used :**
  1. getText( ) method
  2. parse……( ) method
  3. setText( ) method
  4. JOptionPane.showMessageDialog( ) method

getText( ) method – used to obtain text from a GUI component

- This method **returns the text currently stored in a text based GUI component**.
- Components that support getText( ) method include : TextField, Text Area, Button, Label, CheckBox, and RadioButton.

e.g. Consider the following GUI:



- To obtain text from T1 text field, we write : **T1.getText( ) ;**

- The **getText( )** returns a value of **String** type, so we must store the value returned by getText( ) in String type variable. Thus complete statement to obtain text from T1 field would be :

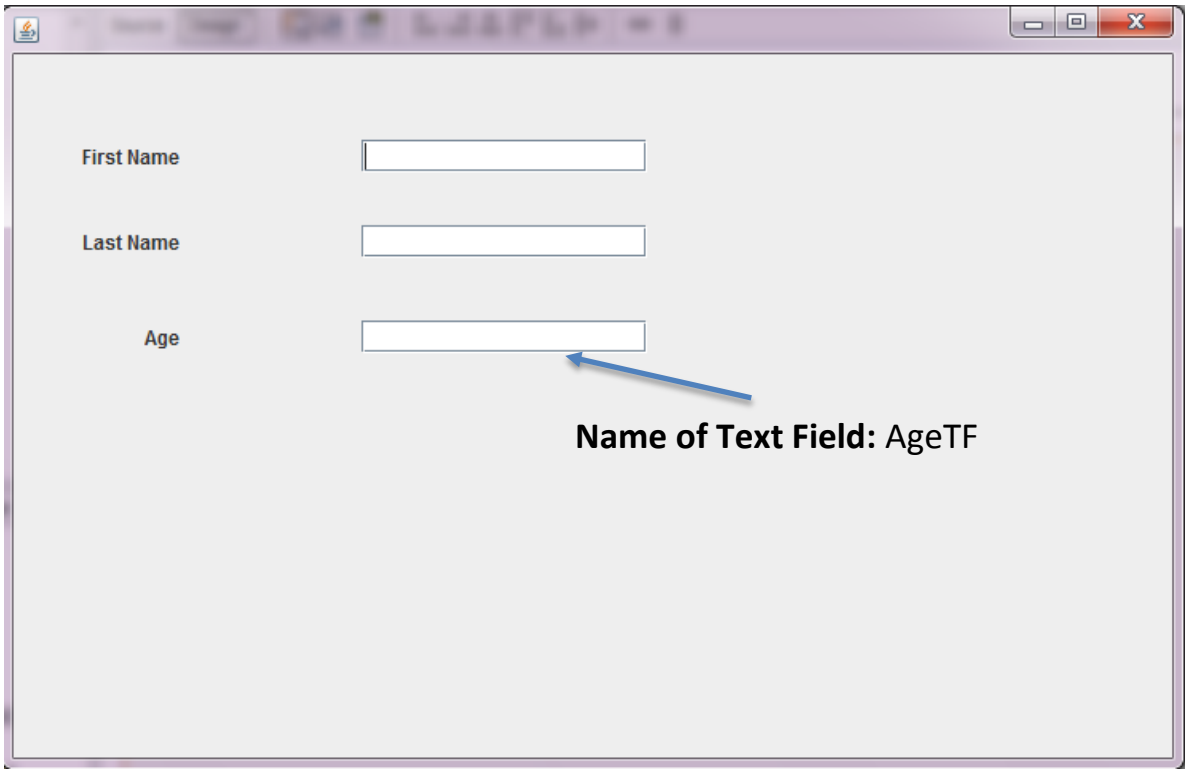
String str1 = T1.getText ( ) ;

- Similarly to obtain text from T2 textfield, we write :

String Str2 = T2.getText( ) ;

parse……( ) methods – Obtaining numbers from a GUI component.

- Sometimes, we use text type components in a GUI but we intend to use it for obtaining numeric values e.g. we may want to read ***age*** of a person through a text field.



- **Since a text field will return text, i.e. String type of data**, you need a method that helps you extract/convert this textual data into a numeric type. For this, **parse( )** methods are useful.

- **There are many parse…….( ) methods that help you parse string into different numeric types. These are :**

(i) Byte.parseByte (String s) converts a String s into a byte type value.
(ii) Short.parseShort (String s) converts a String s into a short type value.
(iii) Integer.parseInt (String s) converts a String s into an int type value.
(iv) Long.parseLong (String s) converts a String s into a long type value.
(v) Float.parseFloat (String s) converts a String s into a float type value.

(vi) Double.parseDouble (String s) converts a String s into a double type value.

- Consider GUI given above, If we want to obtain input from AgeTF textfield, in numeric, say **int ,** form, we need to do it in two steps :
  - First, we have to obtain text from AgeTF by typing a statement like:

String a = AgeTF.getText ( );

  - Then, we need to parse the **String a** obtained above into an **int** by typing statement like :

int cl = Integer.parseInt (a);

- The above two steps can be combined into one also,

int cl = Integer.parseInt (AgeTF.getText( ));

- Similarly to obtain a float value, we may use

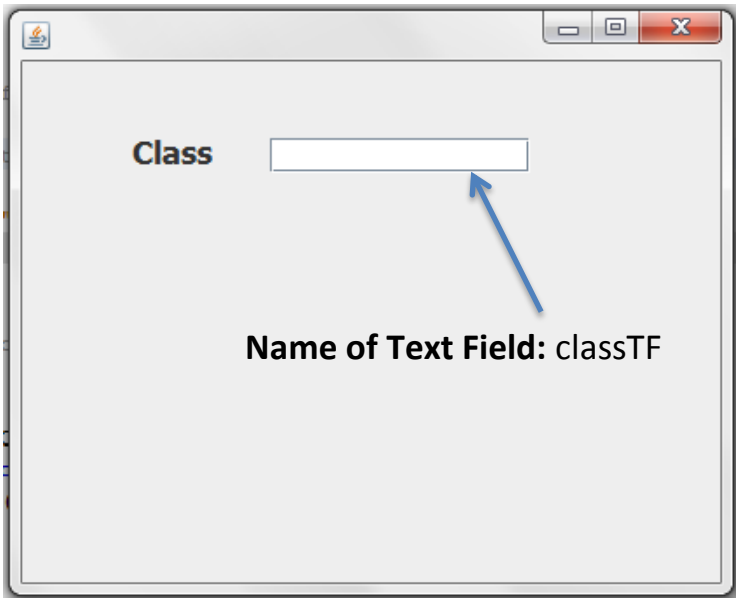
Float.parseFloat(<text obtained from field>);

To obtain a long value, we may use

Long.parseLong(<text obtained from field>);

setText( ) method - Storing text into a GUI component

- This method changes text in a GUI component. The Swing components that support setText( ) method include : TextField, TextArea, Button, Label etc.



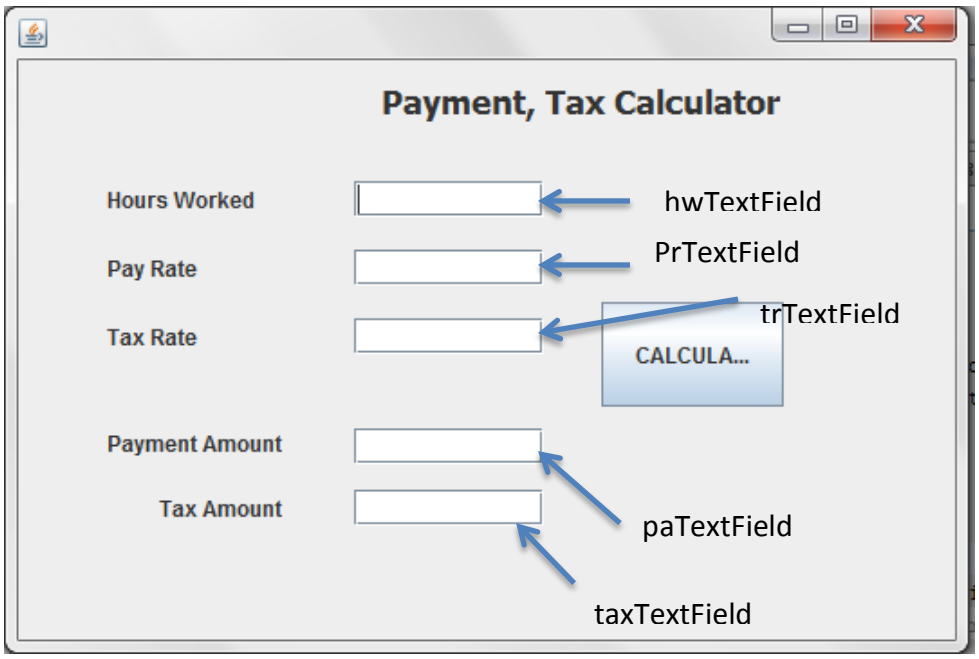
- Suppose we want to change the content of textfield ***classTF*** given above, to "XI", through a code statement; then we use setText( ) method as :

ClassTF.setText("XI");

- The ***setText( )*** changes the value of field before the dot(.) **with the string in its parenthese.**

Example

Design an GUI application that obtains three values in three text fields from user : Hours Worked, Pay Rate and Tax Rate. It should then compute and display Payment Amount (Hours Worked X Pay Rate) and Tax Payable (Payment Amount X Tax Rate) in labels. Assume any numeric data types for these values. The outlook of application should be as shown below :



```
int hours  = Integer.parseInt(hwTextField.getText( )) ;

int prate = Integer.parseInt(prTextField.getText( ));

int trate  = Integer.parseInt (trTextField.getText( ));

int payAmt = hours  * prate ;

int taxAmt  = PayAmt * trate ;

paTextField.setText( " "+payAmt);

txtamtLabel.setText(" "+taxAmt);
```

JOptionPane.showMessageDialog( ) method - Displaying message in a dialog form

- Using this method, we can produce a basic dialog displaying a message to the user. The user will see your message with only an "OK" button to close the dialog.
- **Two steps to use this method** are :

  (a) Firstly, in the source code editor, where you type your code, at top most position type the following line:

  import javax . swing . JOptionPane ;

  (b) Now display desired message as per the following syntax:

  JOptionPane.showMessageDialog (null, "desired-message");

  E.g. to display a message "Hello there!!!", you will write:

  JOptionPane.showMessageDialog (null, "Hello-there!");

CONSTANTS

- Constants are the values that is not going to change when the program is executed. This can be done as :

final double TAXRATE = 0.25 ;

The **keyword final makes a variable as constant** i.e., whose value cannot be changed during program execution.

- Once declared constants, their value cannot be modified e.g. after declaring constant TAXRATE, if we issue a statement like :

TAXRATE = 0.50 ; → Error

It will cause an error, as the value of constants cannot be modified.

OPERATORS IN JAVA

- The ***operations (specific tasks)*** are represented by **operators** and the ***objects of the operation(s)*** are referred to as **operands.**
- **E.g.** In expression **2 + 3** , operands are **2 & 3** and operator is **+** .

TYPES OF OPERATORS:

1. Arithmetic Operators
    - Unary Operators
    - Binary Operators
2. Increment/Decrement Operators
3. Relational Operators
4. Logical Operators
5. Assignment Operators
6. Conditional Operators

ARITHMETIC OPERATORS

- It provides operators for five basic arithmetic calculations: addition, subtraction, multiplication, division and remainder which are:

+, - , * , / and %.

ARITHMETIC OPERATORS

UNARY OPERATORS **BINARY OPERATORS**

UNARY OPERATORS

- Operators that act on one operand are referred to as Unary Operators.

1. **Unary +**

   If a = 5 , then +a means 5

   If b = 4, then +b means 4

2. **Unary –**

   If a = 5, then –a means -5

   If a = -4 , then –a means 4

BINARY OPERATORS

- Operators that act upon two operands are referred to as Binary Operators.

1. **Addition operator (+)**

   e.g. 38 + 42

   a + 5 (where a = 2) results in 7.

   a + b (where a=4, b=6) results in 10.

2. **Subtraction operator (-)**

   e.g. 14 – 3 evaluates to 11.

   a – b (where a = 7, b = 5) evaluates to 2.

3. **Multiplication operator (*)**

   e.g. 3 * 4 evaluates to 12.

   b * 4 (where b = 6) evaluates to 24.

4. **Division operator (/)**

   e.g. 100/50 evaluates to 2.

   a / 2 (a=16) evaluates to 8.

   a / b (where a = 15.9, b= 3) evaluates to 5.3

5. **Modulus operator (%)**

   The % operator produces the remainder of dividing the first by the second operand. For example,

   19 % 6 evaluates to 1, similarly, **17 % 5** evaluates to 2.

Operator + with Strings

- When we use the + with numbers, the result is also a number. However, if we use operator + with strings, it concatenates them, e.g.

  5 + 6 results into 11.
  "5" + "6" results into "56"
  "abc" + "123" results into "abc123"
  " " + 5 results into "5"
  " " + 5 + "xyz" results into _____.

Increment (++) /Decrement (--) Operators

- The operator **++ adds 1** to its operand, and **-- subtracts one**.
- In other words,

  ++a or **a++** is same as **a = a+1.**

  --a or **a--** is same as **a=a-1**.

- Both increment and decrement operators come in **two varieties**:
  1. **Prefix version :** In this, operator comes before the operand, as in ++a or -- a.
  2. **Postfix version :** In this, operator comes after the operand, as in a++ or a --.

The two versions have the same effect upon the operand, but they differ when they take place in an expression.

Working with prefix version (*Principle: Change then use*)

- When an increment or decrement operator precedes its operands, Java performs the increment or decrement operation before using the value of operand.
- E.g. in expression **sum = sum + (++count) ;**
  (assuming initial value of sum and count is 0 and 10)

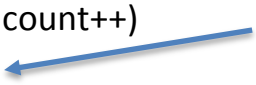
  sum = sum + (++count)
  sum = 0 + 11
  sum = 11

  Value of count is incremented by 1, then used.

Working with postfix version (***Principle: use then Change***)

- When an increment or decrement operator follows its operand, Java first uses the value of the operand in evaluating the expression before incrementing or decrementing the operand's value.
- E.g. in expression **sum = sum + (count++);**
  (assuming initial value of sum and count is 0 and 10)

sum = sum + (count++)
sum = 0 + 10
sum = 10



Value of count is used here, then it will be incremented later.

Q1. Evaluate the expression **x = ++y + 2*y** , if **y = 6**.

Q2. What will be the value of **P=P* ++J** where **J is 22** and **P=3** initially ?

Q3. What will be the value of following, if j =5 initially ?

(i) (5 * ++j) % 6 (ii) (5 * j++) % 6

Q4. What will be the value of **j= --k + 2 * k ,** if **k is 20** initially ?

Q5. Will the value of Y be the same for the two cases given below ?

(i) Y = ++X (ii) Y = X ++ (Given the value of X is 42).

Relational Operators

- Relational operators determine the relation among different operands. Java provides **six relational operators** for comparing numbers and characters.

< (less than)
<= (less than or equal to)
== (equal to)
>(greater than)
>= (greater than or equal to)
!= (not equal to)

- If the comparison is true, the relational expression results into the Boolean value ***true*** and to boolean value ***false***, if the comparison is false. **E.g**.

4 > 3 (true)
3 > 4 (false)
7==0 (false)
3==3 (true)

- **Relational operators don't work with Strings. E.g.**

"123" > "90"
"5" < "3"

Difference between = & == operator

== is equal to operator and is used for comparison. **e.g.**

value == 3, this expression tests whether the value is equal to 3 ?
The expression produce result true if the comparison is true and boolean false if it is false.

= is assignment operator, and is used to assign value to a variable.
E.g.

value = 3, this expression assigns 3 to variable value.

Logical Operators

- Relational operators often are used with logical operators to construct more complex expressions. Three Logical Operators are :
  1. || **(OR Operator)**
  2. && **(AND Operator)**
  3. ! **(NOT Operator)**

The logical OR operator (||)

- The logical OR operator (||) combines two expressions which make its operand. The OR operator evaluates to boolean true if either of its operand evaluates to true. **E.g.**

(4 == 4) || (5 == 8) results into true because first expression is true.

(1 == 0) || (0 > 1) results into false because neither expression is true.

The logical AND operator (&&)

- The logical AND operator (&&) also combines two expressions. The AND operator evaluates to boolean true only if both operands evaluates to true. **E.g.**

(6 == 3) && (4 == 4) results into false because first expression is false.

(6 < 9) && (4 > 2) results into true because both expressions are true.

The logical NOT operator (!)

- The logical NOT operator, written as **!** works on single expression or operand , i.e. it is a unary operator.
- It negates or reverses the truth value of the expression following it. e.g.

  ! (5 > 2) results into false because expression 5 > 2 is true.

!(5>9) results into true because the expression 5 > 9 is false.

Assignment Operator

- The assignment operator , = is used to assign one value to another, e.g.

```
int x , y , z ;
x =9 ;
y = 7;
z = x + y ;
```

Java Shorthand Operators

- Java male use of shorthand operators to simplify certain assignment operators. E.g.

a = a + 10 ;

can be written as

a += 10 ;

The operator += tells the compiler to assign to a the value of a+10. This shorthand *works for all binary operators* in Java.

e.g.

x -= 10 ; is equivalent to x = x – 10 ;
x *= 3 ; is equivalent to x = x * 3 ;
x /= 2 ; is equivalent to x = x/2 ;
x %= z ; is equivalent to x = x % z ;

CONDITIONAL OPERATOR (? :)

- This operator store a value depending upon a condition. This operator is ternary operator i.e. it requires three operands.
- The general form of conditional operator is :

expression 1 ? expression 2 : expression 3 ;

If expression1 evaluates to true, i.e., then the value of whole expression is the value of expression2, otherwise the value of the whole expression is the value of expression3.

e.g.

(marks > =50) ? 'P' : 'F' ;

The identifier ***result*** will have the value 'P' if the expression1 ***marks>=50*** evaluates to true, otherwise ***result*** will have the value 'F'.

Other examples:

(6 > 4) ? 9 : 7 ;

(4 == 9) ? 10 : 25 ;

- The conditional operator can be nested also, i.e. any of the expression2 or expression3 can itself be another conditional operator.

e.g.

(class >= 10) ? (marks >= 80 ? 'A' : 'B') : 'C' ;

The expression first tests the condition if class >=10, if it is true, it tests for another condition if marks >= 80.

Operator Precedence

- It determines the order in which expressions are evaluated. E.g. in following expression :

  y = 6 + 4/2 ;

  the division operation takes place first, rather than addition.

Operator Associativity

- Associativity rules determine the grouping of operands and operators in an expression with more than one operator of the same precedence.
- ***For most operators , the evaluation is done left to right,*** i.e.

  x = a + b – c ;

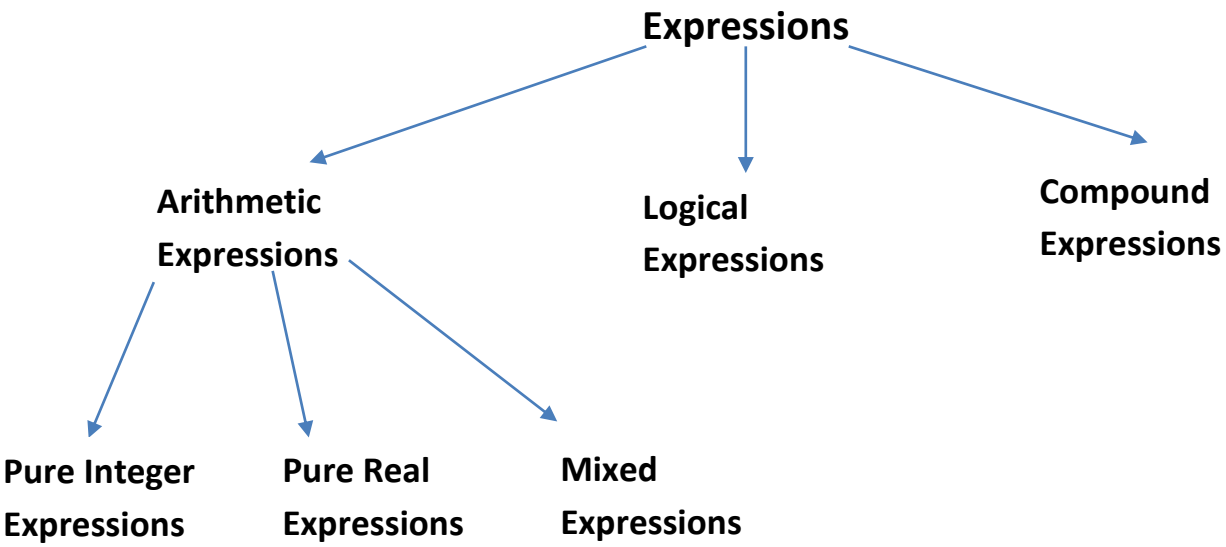
  Here addition and subtraction have the same precedence rating and so a and b are added and then from this sum c is subtracted.

Descending Order of Operator Precedence

Operator	Notes	Associativity
* / %	Multiplication,Division, Modulus	Left to Right
+ -	Addition , Subtraction	Left to Right
< > >= <=	Relational Comparison tests	Left to Right
== !=	Equality	Left to Right
&&	AND Operator	Left to Right
||	OR Operator	Left to Right
? :	Conditional Operator	Right to Left
=	Assignment Operator	Right to Left

Expressions

- An expression in java is any valid combination of operators, constants and variables, i.e. a legal combination of Java tokens.



Arithmetic Expressions

- It may have two or more variables or constants, or two or more expressions joined by valid arithmetic operators. E.g.
  b * c
  (b+9) * (c-d)
  (a+b+c)/d

Pure Integer Expression

- In pure Integer Expression, all operands are of integer type. E.g.
  int a=6, b=8, c=9 ;
  a + b ;
  a + b + c ;
  a * b ;

Pure Real Expression

- In pure Real Expression, all operands are of real(float/double) type. E.g.

  float p=2.4, q=3.6, r=7.2 ;

  a + b ;

  a + b + c ;

  a * b ;

** **pure expressions produce the result having the same datatype as that of its operand. e.g.**

int a=5, b=2 , c ;

a + b will produce result 7 of int type.

a / b will produce result 2 of int type.

Mixed Expression

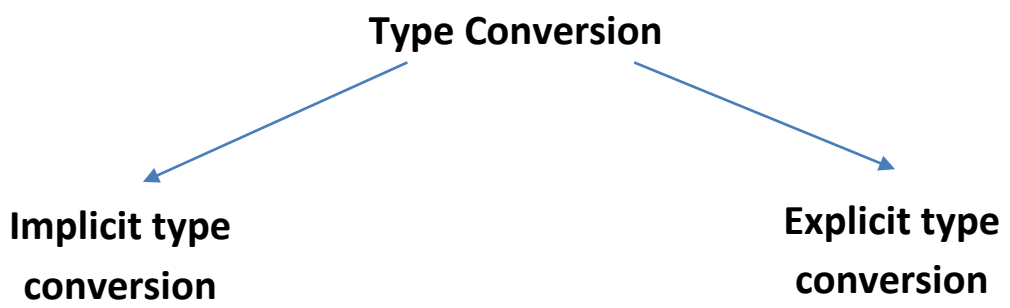
- In mixed expression, the operands are of mixed or different data types. E.g.

```
int z = 9 ;
float c = 5.4 ;
 z  + c ;
```

Mixed Expression

Type Conversion

- When constants and variables of different types are mixed in an expression, they are converted to the same type.
- The process of converting one predefined type into another is called **Type Conversion.**



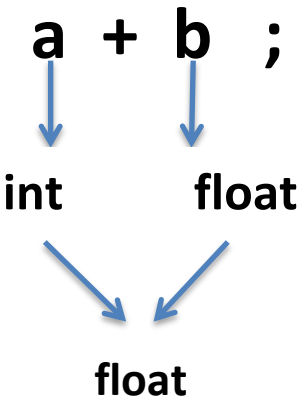
Implicit type conversion

- It is a conversion performed by the complier without programmer's intervention.
- This is done as per the **following type conversion rules** :

- If either operand is of type double, the other is converted to double.
- Otherwise, if either operand is of type float, the other is converted to float.
- Otherwise, if either operand is of type long, the other is converted to long.
- Otherwise both operands are converted to type int.

e.g. int a =3 ;

float b = 3.5 ;



It means that **data type of result** in above expression will be **float.**

Q: Given the following set of identifiers :

```
byte b ;
char ch ;
short  sh ;
int intval ;
long longval ;
float fl ;
```

Identify the datatype of the following expressions:

(a) 'a' – 3

(b) intval * longval – ch ;

(c) fl + longval/sh ;

Explicit Conversion

- This conversion is user defined that forces an expression to be of specific type.
- Explicit conversion is done as shown below :

(type) expression

where ***type*** is a valid data type to which conversion is to be done.

e.g. int a= 2 ; int b = 4.5 ;

a + b ; produces result as 6.5, but in order to obtain result as 6 , we explicitly convert result into int.

To make sure that the expression a + b evaluates to type int, we write it as :

(int) a+ b ;

Logical Expressions (Boolean expressions)

- The expressions that result into false or true are called Boolean expressions or relational expressions. **E.g.**

  (i) x > y
  (ii) (x > y) && (y < z)
  (iii) (x == 3)

Compound Expression

- A compound expression is the one which is made up by combining two or more simple expressions with the help of operator. **E.g.**
  - (a + b) / (c + d)
  - (a > b) || (b > c)

Maths functions available in Java

Functions	Description
pow(x , y)	This function returns x raised to y. (x^y)
exp(x)	This function returns e raised to x (e^x)
log(x)	This function returns the logarithm of x.
sqrt(x)	This function returns the square root of x.
abs(a)	This function returns the absolute value of a.

Q: Write the corresponding Java expression for the following mathematical expressions:

(i) $\sqrt{a^2+b^2+c^2}$

(ii) $2- ye^{2y} + 4y$

(iii) $p + q/(r+s)^4$

(iv) $|e^x - x|$