Index

Sno. Content Page no

1. Token 2
2. Operator 7
3. Operator precedence 11
4. Data types 13
5. Escape sequence 22
6. Type Conversion 23
7. Methods 25
8. Decision making and branching 31
9. Switch Statement 36
10. Looping 38
11. Constructor 40
12. Break 44
13. Continue 45
14. Nested looping 46
15. Scanner 47
16. Math library functions 48
17. String library functions 50
18. Character library functions 54
19. Arrays 56
20. Searching 60
21. Sorting 62
22. Call by value & call by reference 66
23. Two dimensional arrays 69
24. Wrapper Class 72

**Tokens**

Tokens are the various Java program elements which are identified by the compiler. *A token is the smallest element of a program that is* meaningful to the compiler*.* In a Java program, all characters are grouped into symbols called **tokens**.

Tokens supported in Java include:

* + - Keywords
    - Identifiers
    - Literals
    - Operators
    - Comments
    - Separator

**KILOCS**

**EXPRESSION:**

**Assignment** **Punctuator**

**C = A + 5 ;**

**Identifiers** **Operator Literal**

**Types of tokens and their descriptions**

|  |  |
| --- | --- |
| **Token** | **Description** |
| Identifiers | The identifiers are case –sensitive names used by programmers to identify  **variables**,  methods, classes , objects , etc. |
| Keywords | The Keywords are words and names reserved for the Java language. Keywords in appear in  Lowercase and cannot be used as identifiers. |
| operators | The operators are symbols that represent arithmetic , relational , assignment , unary and logical operations. They operate on operands and return results. |
| Separators | The separators are used by the Java compiler to divide the program into segments. Also known as punctuation characters and paired - delimiters |
| Literals | The literals are specific type of data explicitly entered into the code |
| Comments | Comments are non –executable code and it improves readability, understandability and  maintainability of Java programs. |

**KEYWORDS(Reserve words)**

Keywords are identifiers that Java reserves for its own use.

These identifiers have built-in meanings that cannot change.

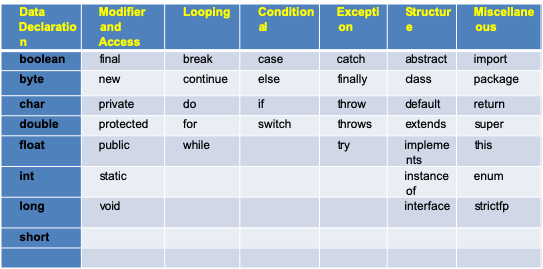
Thus, programmers cannot use these identifiers for anything other than their built-in meanings.

For example , the following statements are not valid .

if =7;

int =6;

**Java Keywords**



**LITERALS**

STRING

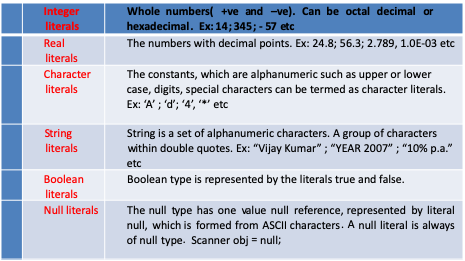
INTEGER

Null

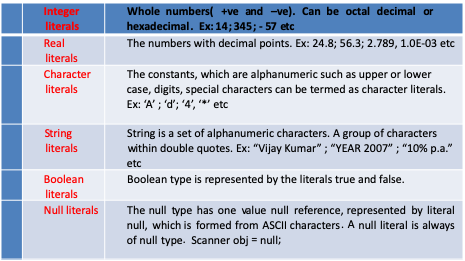
REAL LITERALS

**TYPES OF LITERALS/ CONSTANTS**

CHARACTER

BOOLEAN

**Types of Literals**



**Note: true , false and null are not keywords.**

**Reserved Literals/predefined literals :**

**true, false, null ( We cannot use them as identifiers in the programs.)**

**Identifiers**

Identifier. Identifiers are used by programmers to name things in Java:

things such as variables, methods, fields, classes, interfaces, exceptions, packages, etc.

**Rules of naming an identifier**

* + An identifier can be of any length.
  + It may contain alphabets, digits and underscore ( \_ )and dollar sign ($) characters.
  + They must not be a keyword or true, false or null literal
  + It must not begin with a digit
  + It can start with only 2 special characters $ and \_ Ex: String \_s; or int $n1;
  + Java is case sensitive i.e. uppercase and lowercase letters are treated differently

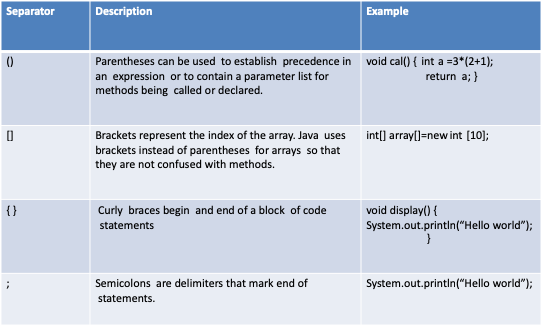
**Separators/ Punctuators**

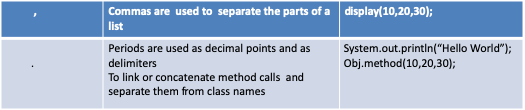
Separators help define the structure of a program . The separators are also known as punctuators . The Java compiler uses separator to break code into segments .

Java uses six types of separators

1. () used to establish precedence in an expression or to contain parameter list for methods being called or declared.
2. { } Used to define the start and end of a block /used to contain the values of arrays.
3. [ ] used to declare array types.
4. ; used to show the end of statements.
5. , used to separate the variables.
6. . used to separate the method or variable from the object

Separators / Punctuator : separators help define the structure of a program. The Java compiler uses separator to break the code into segments.





**OPERATOR**

An operator, in Java, is a special symbols performing specific operations on one, two or three operands and then returning a result.

**Operators:** Operators are special symbols that are commonly used in expressions.

NOTE:

Expressions are statements that return a value

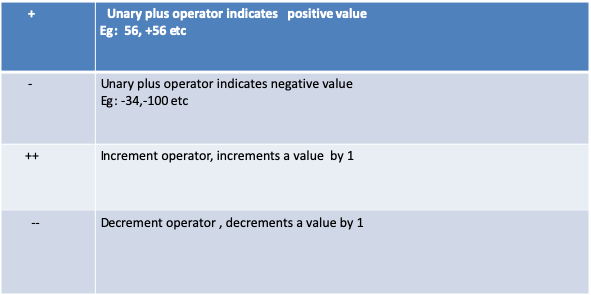
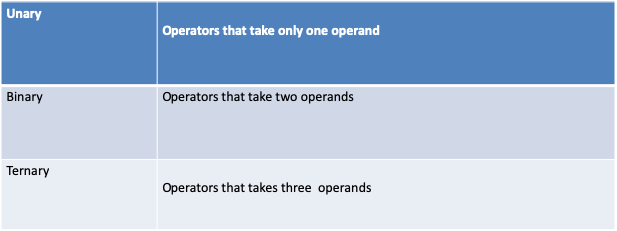
**Operators** and **operands**

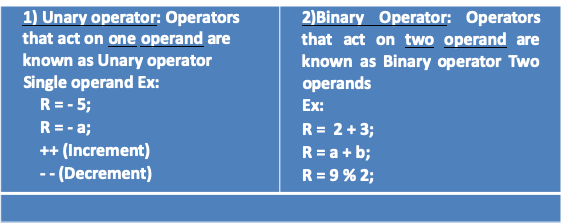
C = a + b \* d

a, b and d are the operands

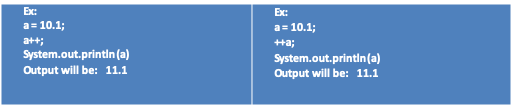
where as +, \* and = are the operators

**CLASSIFICATION OF OPERATORS:**

Operators may be classified by their number of operand



**Working with ++ and -- operator  
Method 1: USED INDEPENDENTLY**



**Method 2:   
While assigning the value**

int a=20;

B = a ++; B = ++ a ;

System.out.println(B); System.out.println( B);

System.out.println(a); System.out.println( a );

Try: B = a -- B = --a

**PRE INCREMENT In an expression:**

sum = 15;

a = 5;

sum =sum + (++a);

**POST INCREMENT In an expression:**

sum = 15;

a = 5;

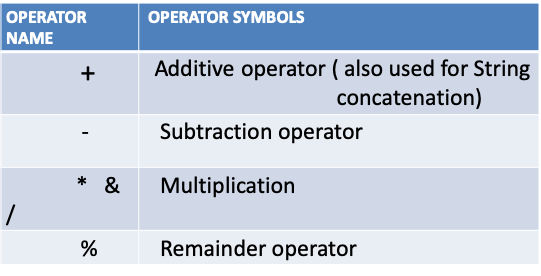
sum =sum + (a++);

NOTE :

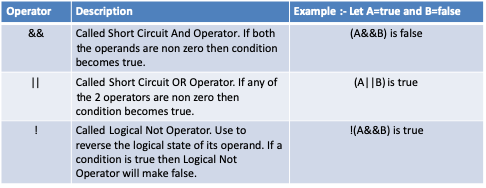
Post Decrement/Pre Dcerement

**BINARY OPERATORS  
  
There are 3 types of operators:**

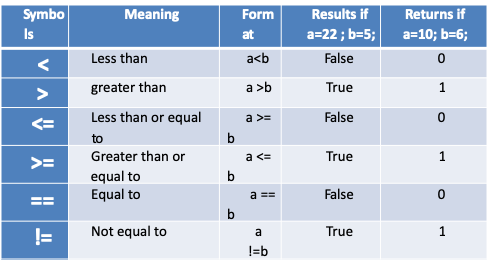
1. Arithmetic operator
2. Relational operator
3. Logical operator

**ARIT HMETIC OPERATORS**

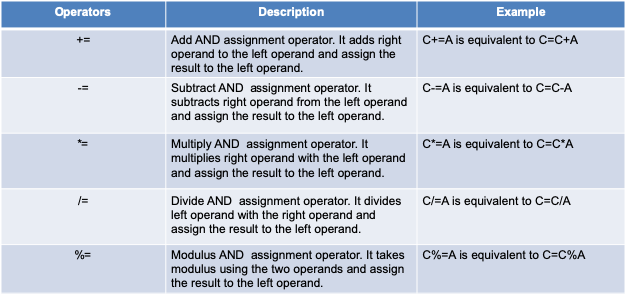
**LOGICAL OPERATORS**

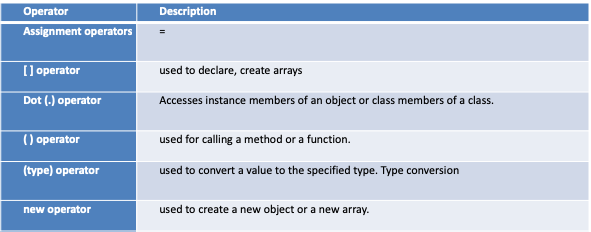


**RELATIONAL OPERATORS**



**SHORT AND ASSIGNMENT OPERATOR**

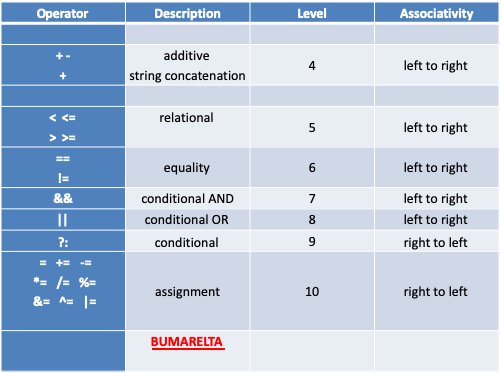
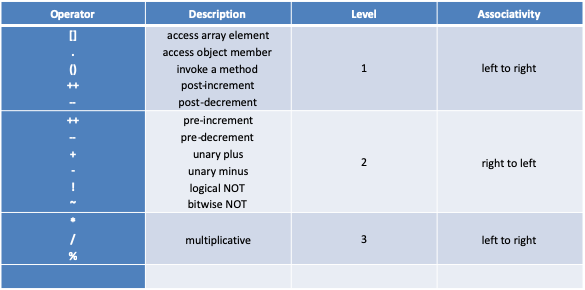


**OTHER OPERATORS**

**WHAT IS OPERATOR PRECEDENCE?**

Operator Precedence determines the order in which expressions are evaluated

EXAMPLE:

Y= 6+4/2 is treated as 6+(4/2) since division has higher precedence than addition 

Consider y= 6+ 4/2

The value of  ***y***can end up being 5 or 8

The / has higher precedence than + . Hence the expression 4/2 is evaluated first and then 6+2 is evaluated .

Answer is 8

***Associativity & Precedence of Operators***

When two operators have the same precedence, associativity determines the evaluation order.

Example : If an expression contains both \* and / .

It has precedence . In this case , associativity determines the evaluation order.

Associativity can be either left to right or right to left.

For example

x = y = z = 17 is treated as x = (y = (z = 17)), leaving all three variables with the value 17,

since the = operator has right-to-left associativity

(and an assignment statement evaluates to the value on the right hand side).

On the other hand, 72 / 2 / 3

is treated as (72 / 2) / 3 since the / operator has left-to-right associativity.

**Expression in Java**

An expression is combination of literals , operators , variables and parentheses used to calculate a value .

**Mixed Expression**

An expression, which includes **different types** of variables or values to yield a result, is known as Mixed expression. Ex:

int a; float b; double d;

double x = a + b \* d

a+b\*d contains different types of variables as int, float and double. – Such expression is called Mixed expression.

**Compound Statements**

Multiple statement written within braces { }, are known as Compound statements.

Eg

void find()

{ a =a+10;

b=b+14;

c=a+b;

}

Types of data

Primitive data

Non Primitive data

Non numeric

Numeric

Integer

Decimal

double

character

float

boolean

byte

int

short

long

arrays

class

BBSILFD

**Primitive types**

Data types are defined as data storage format that a variable can store to perform a specific operation.

The size of the variable and a constant are determined by data types.

* A Data type is defined as the set of possible values a variable can hold.
* It is the means to identify the type of data and associated operations of handling it.

**Integers (numeric integral type):**

* + It stores Integers i.e. it can hold only whole numbers.
  + It can be either positive or negative numbers. **Commas cannot appear.**
  + There are four numeric integral types in Java

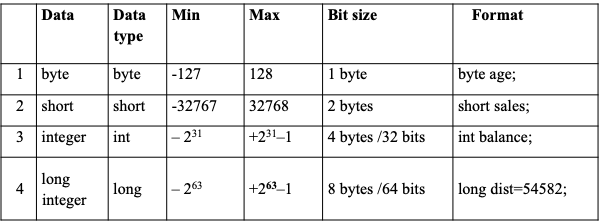
1) byte

2) short

3) int

4). Long

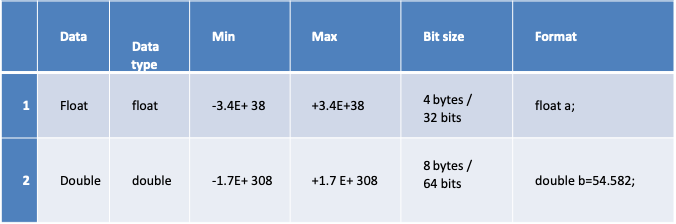
***Primitive types and non-primitive types with examples***



**Floating type/Floating numbers:**

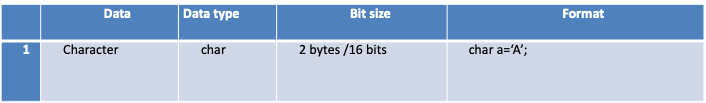
* It can hold only decimal numbers and not integers. i.e. 45.6 or 45.0 but not 45
* It can be either positive or negative numbers.
* Commas cannot appear.
* Can be written in fractional form ( 45.6) or exponent for 45.6E+7 or – 45.6E- 4)
* There should be a number before the decimal and after the decimal point.

**Types of floating type**



**Character type:**

* + It can store only single characters enclosed with the single quotes.
  + Range à 0 - 65536(i.e.216).
  + **We can store 65536 different types of characters in many languages**
  + **JAVA character set is therefore called as Unicode character set ASCII** à **A –Z (65 -90) and a-z (97 -122)**



**Boolean type**

* + - * It is used to represent a single true/false value.
      * **Java reserves 8 bits but uses only 1 bit.**
      * Used for logical results withif.

Ex:

boolean value = true;

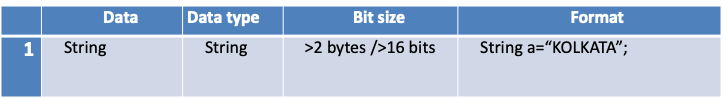
boolean value1 =false; **Note : it does not contain value 0 and 1**

**String class**

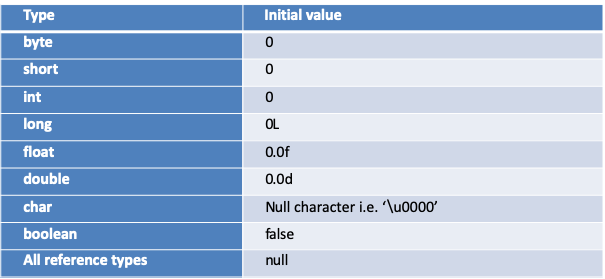
To store a group of characters we use variable /object of String class and

enclose with double quotes.

Eg: String same= “KOLKATA”;

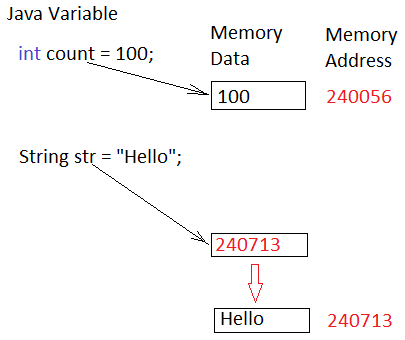
String value =“”; String value1 = “ KA 04 4579”; 

**Default value: A value which a variable assumes by itself if no value is explicitly assigned to it.** Default values of the primitive data types



**What is a variable ?**

A variable is a container that holds value that are used in the program.

The variables provides mechanism to storedata inside a program.

**Comments**

* Comments allow us to place any form of documentation inside our Java code.
* Comments are ignored by the compiler.

Java includes three different types of comments.

**Single line comment:** begins with **//** and continues until the end of the line.

**Multi line comment:**  begins with **/\*** and continues (possibly over many lines) until **\*/** is reached.

**Documentation comment:** begins with **/\*\***It looks like multi line comment. It is a special type of comment used to generate external documentation about the source code. **\*/**

**Dynamic Initialization**

Java allows variables to be initialized dynamically, using any expression valid at the time the variable is declared.

Following program demonstrate dynamic initialization clearly:

// Demonstrate dynamic initialization.

class DynamicInit

{  
public void main( )

{  
double a = 3.0, b = 4.0;

// c is dynamically initialized

double c =a \* a + b \* b;

System.out.print(“Result is " + c);  
} }

Output:

Result is 25

Explanation of **above program**

Three local variables a, b, and c are declared.

 First two, a and b, are initialized as 3.0 and 4.0

 c is initialized dynamically which means declaration and calculation together.

**Difference between constants and variables**

The **difference between variables** and **constants is** that **variables** can change their value at any time but **constants** can never change their value. (The **constants** value **is** locked for the duration of the program).

**Constants** can be very useful, Pi for instance **is** a good example to declare as a **constant**.

void main(int r)

{ final double pi= 3.14; // final is a keyword to make a value constant

System.out.print(pi\* r\*r);

}

Error will occur If you will try to change the value of pi(variable with final keyword)

**What is a character set in JAVA?**

Character set is a set of valid characters. A character represents any letter, digit or any other sign.

* Java uses 16 bits **Unicode** character set to represent the character data.
* This is the global standard for character encoding.
* It supports 65,536 unique characters.

Each Unicode character starts **‘\u’** followed by four hexadecimal digits.

Ex: Java uses UNICODE character set Symbol Description

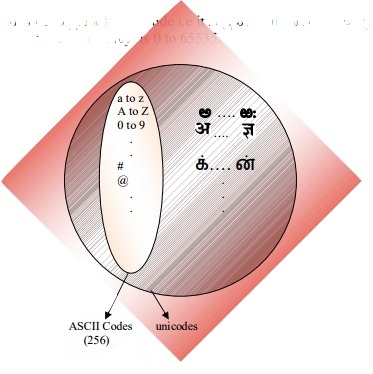
\ u00AE. © copyright symbol

\u00BE. ¾ The fraction

\u03C0 Pie sign

java provides support for Unicode i.e it supports all world wide alphabets. Hence the size of char in java is 2 bytes. And range is 0 to 65535.

Unicode represents most written languages in the world while ASCII does not

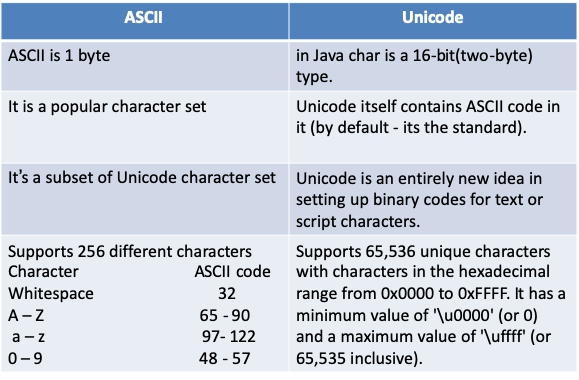
**A**American[](https://i.stack.imgur.com/loX9M.jpg)

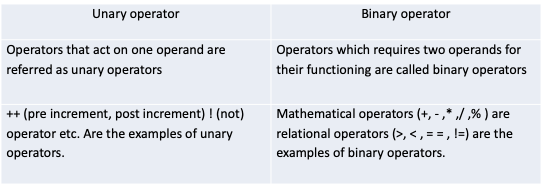
**S** standard

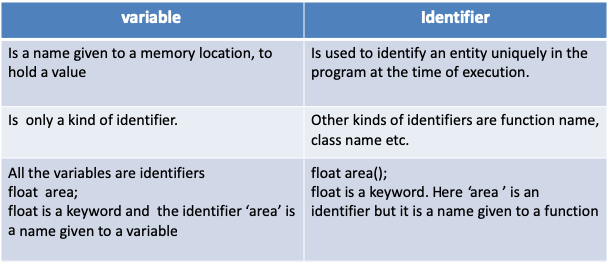
**C** code for

**I** information

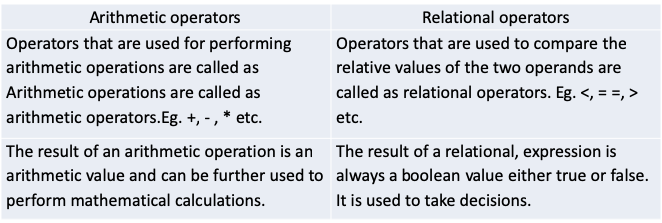
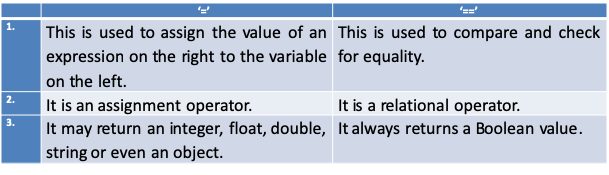
**I** interchange









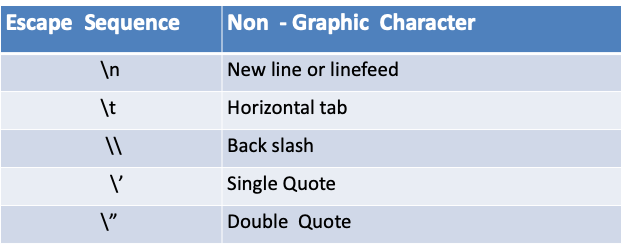




**Escape sequence:**

A character followed by a backslash (\) is called an Escape Sequence. Java allows you to have certain non graphic characters (which cannot be typed directly from keyboard. E.g. backspace, tabs, carriage return, newline etc

These nongraphic characters can be represented using escape sequences

**LIST OF IMPORTANT Escape Sequences**

Example:

All escape sequences are treated as a single character .

To print the sentence : She said “Hello!” to me.

You would write :

System.out.println("She said \"Hello!\" to me.");

Output:

class trial

{

public static void main(String agrs[]) {

String a=" Java is programming language \n developed \n by \t \'James Gosling\' ";

System.out.println(a );

}}

**Type conversion**

The process of converting one predefined to another is called as Type conversion

**Implicit conversion** /Coercion /Widening -> is done by the compiler where the smaller data types is promoted into higher data types. This is called as Type promotion

The following conversion takes place implicitly:

byte->short-> int->long->float->double

char->int->long->float->double

In some cases we might want to assign value of one type to variable of another type.

If both the source and destination types are compatible, then java performs the

conversion **implicitly(automatically).**

Java automatically converts from one type to another only when the following two

conditions are satisfied:

1) Both types are compatible with each other.   
2) The size of destination type is more than the source type.

•int can be auto converted to long since the size of long is 64, where as size of int is 32.

Similarly a byte can be auto converted or implicitly converted to int.   
•int can also be implicitly converted to float and double

**Example:** In the given code 4 bytes integer value is assigned to 8 bytes double value.

int. x = 10; // occupies 4 bytes

double y = x; // occupies 8 bytes

System.out.println(y); // prints 10.0

In the above code 4 bytes integer value is assigned to 8 bytes double value.

**Explicit Conversion**

Explicit conversion is a forced conversion done by the programmer using (type) operator This process is called as Type casting . Also known as **forced conversion**.

A data type of higher size (occupying more memory) cannot be assigned to a data type of lower size. This is not done implicitly by the JVM and requires **explicit casting**; a casting operation to be performed by the programmer. The higher size is narrowed to lower size.

(typename)value

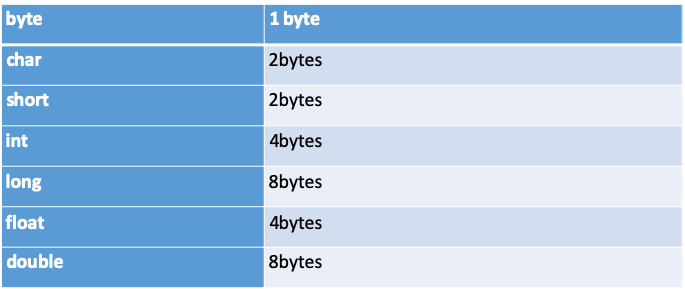
float b = (float) (x+y/2)

double x = 10.5;

int y = (int) x;

The double x is explicitly converted to int y.

The thumb rule is, on both sides, the same data type should exist.



B

C

S

I

L

F

D

**Methods**

It is a set of commands that can be used over again. Executable instructions are methods, functions, subprograms or subroutines.

**Why methods are needed?**

To make the program easy and understandable and it also looks neat and organized.

## Creating a Method:

Actions performed by objects are specified through functions/methods. In general, a method has the following syntax:

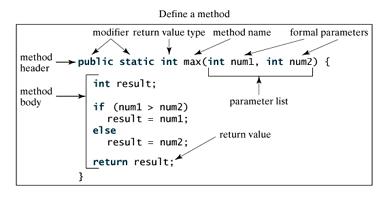
modifier returnValueTypemethodName(list of parameters){

// Method body;

}

A method definition consists of a method **header/prototype** and a method body. Here are all the parts of a method:

* **Modifiers:** The modifier, which is optional, tells the compiler how to call the method. This defines the access type of the method.
* **Return Type:** A method may return a value. The returnValueType is the data type of the value the method returns. Some methods perform the desired operations without returning a value. In this case, the returnValueType is the keyword void. Return type is void if you do not want the method to give you any data back. If you add two integers and want the results of that integer, your return type would be int.
* **Method Name:** This is the actual name of the method. The method name and the parameter list together constitute the method **signature.**
* **Parameters:** A parameter is like a placeholder. When a method is invoked, you pass a value to the parameter. This value is referred to as **actual parameter or argument**. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.
* **Method Body:** The method body contains a collection of statements that define what the method does.



## The void Keyword:

This section shows how to declare and invoke a void method. Following example gives a program that declares a method named printGrade and invokes it to print the grade for a given score.

## Example:

publicclassTestVoidMethod{

publicvoid main(){

printGrade(78.5);}// calling a method

publicvoidprintGrade(double score){

if(score >=90.0){

System.out.println('A');}

elseif(score >=80.0){

System.out.println('B');}

elseif(score >=70.0){

System.out.println('C');}

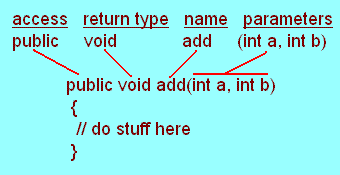
else{

System.out.println('F'); } } }

This would produce following result:

C

Here the printGrade method is a void method. It does not return any value. A call to a void method must be a statement. So, it is invoked as a statement in line 3 in the main method. This statement is like any Java statement terminated with a semicolon.



Since the return type is void, you will have to write what you want the method to do, inside the method such as printing.

**Return Statement:** The statement which sends back the value/result from a method to the caller program is known as Return statement. Basically it is used at the end of a function, which can be referred as function terminator.

Return statement is useful in two ways: First, an immediate exit from the function is caused as soon as a return statement is encountered. Secondly, used to return a value to the calling code.

Syntax: return <value>

E.g. { int s =m + n;

return (s);}

Features of Return statement

1. It is applied at the end of a method, from where it terminates.
2. The return type of the function must be compatible to the value returned from the function.
3. No statement in the method can be executed after return statement.

E.g. { int s =m + n;

return (s);

int p =m\* n; }

1. It can only return a single value from a method to its caller.

E.g.{ int s =m + n;

int p=m\*n;

return (s, p);} //invalid

## Calling/Invoking a Method:

In creating a method, you give a definition of what the method is to do. To use a method, you have to call or invoke it. There are two ways to call a method; the choice is based on whether the method returns a value or not.

When a program calls a method, program control is transferred to the called method. A called method returns control to the caller when its return statement is executed or when its method-ending closing brace is reached.

If the method returns a value, a call to the method is usually treated as a value. For example:

int larger =max(30,40);

**Actual and Formal parameters**

The values are passed with the defined parameters in a function.

//Defining Method

Return type Formal Parameters

int mul(int a, int b)

{

return a\*b;}

void main()

{ int l = 5;

int w = 3;

int area= mul ( l, w);

}

Actual Parameters

**Formal Parameters** are the ones that appear in function definition.

**Actual Parameters** are the parameters appearing in function call statement.

**Different ways of defining a function**

1.Receiving values but not returning outcome to the caller

public void add(int a, int b) //parameterized method

{

System.out.println(a+b); } // no return type

2. Receiving values and returning outcome to the caller

public int add(int a, int b) //parameterized method

{

return (a+b); } // return type

3. Neither receiving values nor returning outcome to the caller

public void add() //non parameterized method

{ int a=5,b=6;

System.out.println(a+b); } // no return type

## Passing Parameters by Values:

When calling a method, you need to provide arguments, which must be given in the same order as their respective parameters in the method specification. This is known as parameter order association.

Pass by value is the process of passing a copy of actual parameters to the formal parameters. Any changes made in the formal parameters does not reflect on the actual parameters.

For simplicity, Java programmers often say passing an argument x to a parameter y, which actually means passing the value of x to y.

Method can be called by inserting the two integers that will go into the method parentheses :

add(7,4); //calling a method

If we want to use the results for something else within our main method, we will need a return type. Since we are adding integers, int return type is adequate:

public int add(int a, int b) //return type function

{//return type required }

Since we have an integer return type, we need to have a return statement that will return an int.

We can use a local variable ( c ) to hold the result and return the value of that local variable.

Note: c must be the same data type as the return type – which is int in this example:

public int add(int a, int b)

{ int c =a+b;

return c;}

This can also be shortened by not using the local variable:

public int add(int a, int b){

return a+b;}

In order to use the above with the int return type, in the main method you can either store the result in a variable:

int temp = add(5,7);//calling the method and storing the returned value in temp variable

or you can print out the result by calling it in your print statement:

System.out.println(add(5,7));

## The Scope of Variables:

The scope of a variable is the part of the program where the variable can be referenced. A variable defined inside a method is referred as a **local variable**.

The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be declared before it can be used.

A parameter is actually a local variable. The scope of a method parameter covers the entire method.

**Method Overloading**

When more than one functions have same name but the type and number of arguments are different, they are said to be overloaded. Thus , parameter list of each function should be unique. Function overloading implements the concept of polymorphism.

Example:

class overloading

{ public void sum(double a, double b)

{System.out.println(“sum of float values :” +(a+b)); }

public void sum(int a, int b)

{System.out.println(“sum of int values :” +(a+b)); }

public void sum(int a , int b, int c)

{System.out.println(“sum of int values :” +(a+b+c)); }

}

**Decision making and branching**

**Introduction:**

Generally a program executes it's statements from beginning to end. But not many programs execute all their statements in strict order from beginning to end. Most programs decide what to do in response to changing circumstances.  
In a program statements may be executed sequentially, selectively or iteratively..

When a program breaks the sequential flow and jumps to another part of the code, it is called **selection or branching.**   
When the branching is based on a particular condition, it is known as **conditional branching.**   
If branching takes place without any condition, it is known as **unconditional branching.**  
  
Java language supports two types of selections statements: **if and switch.** In addition, in certain circumstances '?' operator can be used as an alternative to if statements.

**DECISION MAKING WITH IF STATEMENT:**

An if statement tests a particular condition; if the condition evaluates to true , a course of action is followed i.e a statement or set of statements is executed . Otherwise (if the condition evaluates to false) ,the course-of-action is ignored.  
The if statement may be implemented in different forms depending on the complexity of conditions to be tested.

* Simple if statement
* if....else statement
* Nested if....else statement
* else if ladder

**SIMPLE IF STATEMENT:**

The **syntax** of the if statement is shown below:

if(expression)  
statements;  
  
Where a statement may consist of a single statement , a compound statement, or nothing. The expression must be enclosed in parentheses. If the expression evaluates to true, the statement is executed, otherwise ignored.

Following figure illustrates the if construct of java:  
﻿

|  |
| --- |
| [https://lh5.googleusercontent.com/-g94vyqsThRc/TXj4m8S5YJI/AAAAAAAAACE/Pw5Z_3l4Gew/s640/If+construct.jpg](https://lh5.googleusercontent.com/-g94vyqsThRc/TXj4m8S5YJI/AAAAAAAAACE/Pw5Z_3l4Gew/s1600/If+construct.jpg) |
| Flow Chart of IF control |

﻿**Example:**  
if(a>10)  
if(b<15)  
c=(a-b)\*(a+b);  
  
If the value of a is greater than 10, then the following statement is executed which in turn is another if statement. This if statement tests b and if b is less than 15, then c is calculated.  
  
**THE IF....ELSE STATEMENT:**  
This form of if allows either-or condition by providing an else clause. The syntax of the if-else statement is the following:  
  
if(expression)  
  statement 1;  
else  
  statement 2;  
  
If the expression evaluates to true i.e., a non-zero value, the statement 1 is executed, otherwise statement 2 is executed. The statement 1 and statement 2 can be a single statement, or a compound statement, or a null statement.  
  
Following figure illustrates the if else construct of java:  
﻿

|  |
| --- |
| [https://lh6.googleusercontent.com/-pm91qkfDIfU/TXj7QOLZgZI/AAAAAAAAACI/x6-N2fwkJ7g/s640/if+else+control.jpg](https://lh6.googleusercontent.com/-pm91qkfDIfU/TXj7QOLZgZI/AAAAAAAAACI/x6-N2fwkJ7g/s1600/if+else+control.jpg) |
| Flow chart of if-else control |

﻿**Example:**  
int a,b;  
  
if(a<=b)  
   a=0;  
else   
   b=0;  
  
**NESTED IF....ELSE STATEMENT:**  
A Nested if is an if that has another if in it's 'if's' body or in it's else's body. The nested if can have one of the following three forms:  
  
1.    if(expression1)  
        {  
          if(expression2)  
           statement-1;  
         else  
           statement-2;  
        }  
      else  
         body of else;  
  
2.     if(expression1)  
        body-of-if;  
       else   
       {  
          if(expression2)  
           statement-1;  
         else  
           statement-2;  
        }  
  
3.      if(expression1)  
        {  
          if(expression2)  
           statement-1;  
         else  
           statement-2;  
        }  
      else  
     {  
       if(expression3)  
           statement-3;  
      else  
         statement-4;  
     }  
  
**THE ELSE-IF LADDER:**  
A common programming construct in the java is the if-else-if ladder , which is often also called the if-else-if staircase because of it's appearance.  
  
It takes the following general form:  
if(expression1) statement1;  
  else  
      if(expression2) statement2;  
  else   
      if(expression3) statement3;  
   :  
   else statement n;  
  
The expressions are evaluated from top downward. As soon as expression evaluates to true, the statement associated with it is executed and the rest of the ladder is bypassed. If none of the expression are true, the final else gets executed. If the final else is missing, no action takes place if all other conditions are false.  
  
Although it is technically correct to have indentation in the if-else-if ladder as shown above. however, it generally leads to overly deep indentation . For this reason, the if-else-if ladder is generally indented like this:  
  
if(expression1)  
  statement 1;  
elseif(expression2)  
  statement 2;  
elseif(expression3)  
  statement 3;  
    :  
else   
  statement n;  
  
  
**THE ?: OPERATOR (Ternary Operator) :**  
Java has an operator that can be used as an alternative to if statement. You are familiar with this operator, the conditional operator **? :** This operator can be used to replace if-else statements of the general form:  
  
     if (expression 1)  
        expression 2;  
     else  
         expression 3;  
  
The above form of if can be alternatively written using **?:** as follows:  
  
       expression 1? expression 2: expression 3;

Difference between ternary and If- else statement

|  |  |
| --- | --- |
| Ternary operator | If-else statement |
| Gives only one value | Is flexible and can have more than one statement in a block |
| This is more concise , clean and compact but in its nested form it is more complex. | Nested form is not complex |

**THE SWITCH STATEMENT**

Java provides a **multiple branching selection statement** known as switch. This selection statement successively tests the value of an expression against a list of integer or character constants. When a match is found, the statements associated with that constant are executed.

The general form of switch is:

switch(expression)

{. case value1:

Code segment 1; break;

case value2:

Code segment 2; break;

...

case valuen:

Code segment n; break;

default:

default Code segment. }

A switch statement is used for multiple way selection that will branch to different code segments based on the value of a variable or an expression . The optional default label is used to specify the code segment to be executed when the value of the variable or expression does not match with any of the case values. if there is no break statement as the last statement in the code segment for a certain case, the execution will continue on into the code segment for the next case clause without checking the case value.

**The FALL THROUGH(missing break statement)**

* When the switch variable is equal to a case, the statements following that case will execute until a break statement is reached.
* When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
* Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.

**TERMINATING THE PROGRAM**

* Java provides a feature to terminate the currently running program by using exit() function of class system.
* System.exit(n); here n serves as status code.
* A non\_zero status code indicates abnormal termination.

Eg: Write a menu driven program to print square or cube of the number depending upon the user’s choice(S for square , C for cube)

class swtch\_prg

{ void main(int num , char alp)

{ switch(alp)

{ case ‘S’: System.out.println(“square =“+ num\*num); break;

case ‘C’: System.out.println(“cube =“+ num\*num\*num); break;

default: System.out.println(“invalid choice”); System.exit(0); } } }

**Difference between IF and switch:**

|  |  |
| --- | --- |
| Can test relational and logical expression | test only equality condition |
| Range can be tested. | Range cannot be tested |
| The value of more than one variable or expression can be compared to the value of more than one variable or expression | The value of the variable or of an expression is tested for equality against possible constants. |
| Floating point tests can be performed | Floating point tests cannot be performed |

|  |  |
| --- | --- |
| break | System.exit(0) |
| 1. break is keyword | exit() is a library function |
| 2. It causes immediate exit from the current loop or block | exit(0) used to terminate program execution. |

**ITERATION OR LOOPING**

**INTRODUCTION:**  
The iteration statements allow a set of instructions to be performed repeatedly until a certain condition is fulfilled. The iteration statements are also called loops or looping statements.

There may be a situation when we need to execute a block of code several number of times, and is often referred as a loop.

Java has very flexible three looping mechanisms. You can use one of the following:

* while Loop
* do...while Loop
* for Loop

All three repeat a set of statements as long as a specified condition remains true. The specified condition is generally referred to as a **loop control**. For all three loop statements, a true condition is any nonzero value. A zero value indicates a false condition.  
  
**THE WHILE STATEMENT:**  
The most simple and general looping structure available in java is the **while statement.** The syntax of while loop is:  
**while(condition)  
   {  
      // loop-body  
    }**Where the loop body may contain a single statement, a compound statement or an empty statement. The loop iterates while the condition evaluates to true. When the expression becomes false, the program control passes to the line after the loop body code.

When executing, if the *boolean\_expression* result is true then the actions inside the loop will be executed. This will continue as long as the expression result is true.

Here key point of the *while* loop is that the loop might not ever run. When the expression is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

**//program for while loop**  
class whiletest {  
     public void main() {  
             int n=10;  
         while(n>0)  
         {  
           System.out.println("tick", +n);  
           n--;  
          } } }  
   

**THE DO While STATEMENT:**  
Unlike the while loop, the do-while is an exit-controlled loop i.e. it evaluates its text-expression at the bottom of the loop after executing it's loop-body statements. This means that a do-while loop always executes at least once. The syntax of the do-while loop is:

**do {  
      loop-body;  
     }  
while(condition);**

**Example:**  
 Class doWhileCheck {

public void main () {  
              int n=10;  
          do  
            {  
               System.out.println("tick" +n);  
                n--;  
            }  
             while(n>0);  
        } }

**THE FOR STATEMENT:**  
The **for** loop is the easiest to understand of the java loops. All its loop-control elements are gathered in one place (on the top of the loop), while in the other loop construction, they( top-control elements) are scattered about the program. The Syntax of the for loop statement is:

**for( initialization expression(s); test condition; update expression)  
{   
 loop-body;  
}  
//program showing usage of for loop**

class forTest {  
   public void main( ) {  
   int i;  
    for( i=1; i<=10; i++)  
     System.out.println(i); } }

 Here is the flow of control in a for loop:

* The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
* Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement past the for loop.
* After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the Boolean expression.
* The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

**The main difference between for loop, while loop, and do while loop is**

* While loop checks for the condition first. so it may not even enter into the loop, if the condition is false.
* do while loop, execute the statements in the loop first before checks for the condition. At least one iteration takes places, even if the condition is false.
* for loop is similar to while loop except that
* initialization statement, usually the counter variable initialization
* a statement that will be executed after each and every iteration in the loop, usually counter variable increment or decrement

**I Infinite or endless loop**

**publicclass**infiniteloop{

**publicvoid** main(){

**for**(;;)**//no condition**

System.out.println("Hi");}}

**II. Use of comma Operator**

**publicclass**CommaOperator{

**void** main(){

**for**(**int** i =1, j = i +10; i <5; i++, j = i \*2)

{

System.out.println("i= "+ i +" j= "+ j);}}

**III Delay loop**

**publicclass**Delayloop{

**publicvoid** main(){

**for**(int i=1;i<1000;i++) ; **//ending with semicolon**

System.out.println("Hi");}}

**Control variable:** A variable that initialize a value and than incremented or decremented as required is called a control or counter variable.

**Block:** Is a st of instructions enclosed in braces.It begins with an opening curly brace { and ends with closing brace }.

Difference between and While and do while statement

|  |  |
| --- | --- |
| while | do while |
| The statement is not executed when condition is not satisfied. | The loop executes the statement at least once when condition is not satisfied. |
| The condition is tested before execution. | The condition is tested after execution. |
| The loop terminates if the condition becomes false. | If the condition is true, the computer keeps executing the loop. |
| It is known as entry controlled loop | It is called an exit controlled loop |

|  |  |  |
| --- | --- | --- |
|  | for | while |
| 1. | The for statement is used when it is known how many times the loop is to be executed. | The while is used when it is not known how many times the loop is to be executed. |
| 2 | It is programmer controlled | It is user controlled. Depending on the user’s response the loop execution is continued or terminated. |

# **Constructors**

# A constructor is a member function of a class with the same name as that of a class. It is defined like other member functions of a class. A constructor is a special member function which is automatically called when an instance/object of the class is declared .It is used to initialize the object of that class with the legal initial value.

Characteristics of Constructor:

1. A Constructor has the same name as the class.
2. Constructor has no return type not even void.
3. A Constructor is a member method of the class.
4. Constructors are used to initialize the data members of the class.
5. Constructor is automatically called when an object of the class is created.

Types of Constructors:

Default Constructor: The Constructor which do not take any parameter or arguments. If a programmer does not provide any constructor in the class then java automatically supplies a default constructor. Compiler initializes member variables to its default values.

* numeric data types are set to 0
* char data types are set to null character(‘’)
* boolean data types are set to false
* reference variables are set to null

**Assignment 1**: Create default Constructor

classDemo{

       int  value1;

       int  value2;

       Demo(){

          value1 = 10;

          value2 = 20;

      publicvoiddisplay(){

         System.out.println("Value1 === "+value1);

         System.out.println("Value2 === "+value2);}}

Parameterized Constructor: is a Constructor that accepts parameters. It initialize an object with values that are passed upon object creation.

**Assignment 2**: Create Parameterized Constructor

classDemo{

       int  value1;

       int  value2;

       Demo(int value1, int value2){

          this.value1 = value1;

          this.value2 = value2;

         publicvoiddisplay(){

         System.out.println("Value1 === "+value1);

         System.out.println("Value2 === "+value2);}}

# **Constructor Overloading:**

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

class Demo{

       int  value1;

       int  value2;

Demo( ) // default constructor

{value1=value2=0;}

       Demo(int value1, int value2){ //parameterized constructor

          this.value1 = value1;

          this.value2 = value2;

         public void display(){

         System.out.println("Value1 === "+value1);

         System.out.println("Value2 === "+value2);}}

Difference between constructor and method:

|  |  |
| --- | --- |
| METHODS | CONSTRUCTORS |
| Is a member function with different name from that of a class | Is a member function with same name as that of its class |
| Is used to perform some operations | Is to create an object |
| May or may not contain return type | There is no return type given in a constructor. |
| They are used to perform specific task | They are used to initialize the data members |
| It is invoked using the dot operator | It is invoked using the new operator |

**The break Keyword:**

The *break* keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement.

The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

Syntax:

The syntax of a break is a single statement inside any loop:

break;

Example:

publicclassTest{

Publicvoid main(){

for(int x =1; x< 6; x++ ){

if( x > 3){

break;}

System.out.print( x );

System.out.print("\n");

} } }

This would produce following result:

1

2

3

**The continue Keyword:**

The *continue* keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

* In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.
* In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

Syntax:

The syntax of a continue is a single statement inside any loop:

continue;

Example:

public class Test {

public void main() {

for(int x =1; x<6; x++ ) {

if( x == 3 ) {

continue;

}

System.out.print( x );

System.out.print("\n");

} } }

1

2

4

5

Difference between break and continue

|  |  |
| --- | --- |
| Break | Continue |
| It branches the flow of execution to a statement outside the loop. | It branches the flow of execution to next iteration by skipping other statements of loop but updates the control variable (for loop)and checks condition. |

**Nested Loops**

A loop within another loop in known as nested loop.

But in nested loop, the inner loop must terminated before the outer loop. The following is the example :

for( i=1 ; i<5 ;i++)// Outer loop

{

System.out.println(i);

for(j=1;j<=i; j++)//Inner loop

System.out.println(“&”);

}

*:*

Here **’j’** loop is within **‘i’** loop

Output will be:

&

&&

&&&

&&&&

Inner for loop will executed for each value of i . The variable i takes values 1,2,3 and 4. The inner loop is executed i times(i.e. 4 in this example).

General format or Syntax:

for(initializing expression ; condition ; increment/decrement expression)

{ //body of outer loop;

for(initializing expression ; condition ; increment/decrement expression)

{

//Body of inner loop;

}

}

**Scanner class**

The Scanner class is available injava.util package, which allows the user to read values of various types.

import java.util.Scanner;

Class Constructors

There are two constructors that are particularly useful: one takes an InputStream object as a parameter and the other takes a FileReader object as a parameter.

Scanner in = new Scanner(System.in); // System.in is an InputStream

**‘Scanner’**- is the name of the class, ‘in’-Object name, can be any name of your choice.

**‘new’**-keyword is to have dynamic allocation of scanner object.

**Constructor**- is of special type, which automatically invoked to store values in the object from console.

**‘System.in’**-Parameter allows the constructor to receive the data from the keyboard.

Numeric and String Methods

|  |  |
| --- | --- |
| *Method* | *Returns* |
| int nextInt() | Returns the next token as an int. If the next token is not an integer, InputMismatchException is thrown. |
| long nextLong() | Returns the next token as a long. If the next token is not long, InputMismatchException is thrown. |
| float nextFloat() | Returns the next token as a float. If the next token is not a float or is out of range, InputMismatchException is thrown. |
| double nextDouble() | Returns the next token as a double. If the next token is not a double or is out of range, InputMismatchException is thrown. |
| String next() | Finds and returns the next complete token from this scanner and returns it as a string; a token is usually ended by whitespace such as a blank or line break. If not token exists, NoSuchElementException is thrown. |
| String nextLine() | Returns the rest of the current line, excluding any line separator at the end. |
|  |  |

The Scanner looks for *tokens* in the input. A token is a series of characters that ends with what Java calls *whitespace*. A whitespace character can be a blank, a tab character, a carriage return, or the end of the file. Thus, if we read a line that has a series of numbers separated by blanks, the scanner will take each number as a separate token. Although we have only shown four numeric methods, each numeric data type has a corresponding method that reads values of that type.

The next method returns the next input value as a string, regardless of what is keyed. For example, given the following code segment and data

int number = in.nextInt();

float real = in.nextFloat();

long number2 = in.nextLong();

double real2 = in.nextDouble();

String string = in.next();

String string = in.nextLine();

nextLine()reads the *rest* of the line and returns it as a string.

char ch= in.next().charAt(0); // to take charcter

How does scanner reads the data?

Scanner reads the data and breaks down the data into tokens. Tokens are part of data, separated by Delimiters (eg. White space etc.)

Delimiters are to separate the Tokens. By default delimiter is referred to be whitespace.

**import java.util.Scanner;**

**public class scan**

**{Scanner sc=new Scanner(System.in);**

**void main()**

**{System.out.println("enter number");**

**int n=sc.nextInt();**

**System.out.println("enter name");**

**String na=sc.nextLine();**

**System.out.println("enter word");**

**String wr=sc.next();**

**System.out.println("enter character");**

**char ch=sc.next().charAt(0);**

**System.out.println(n+ " "+na+" "+wr+" "+ch);**

**}**

**}**

**LIBRARY FUNCTIONS**

**MATH CLASS**

Number Methods:

|  |  |
| --- | --- |
| **SN** | **Methods with Description** |
| 1 | [abs()](http://www.tutorialspoint.com/java/number_abs.htm) Returns the absolute value of the argument. Eg. 12.3 =12.3 and -23.5=23.5 |
| 2 | [ceil()](http://www.tutorialspoint.com/java/number_ceil.htm) Returns the smallest integer that is greater than or equal to the argument. Returned as a double.eg. 8.2=9.0 and -8.2 = -8.0 |
| 3 | [floor()](http://www.tutorialspoint.com/java/number_floor.htm) Returns the largest integer that is less than or equal to the argument. Returned as a double. eg. 8.2 = 8.0 and -8.2=-9.0 |
| 4 | [rint()](http://www.tutorialspoint.com/java/number_rint.htm) Returns the truncated value of the argument. Returned as a double. Eg.13.9=14.0, -8.2= -8.0 , -9.7= -10.0 and 4.5= 4.0 |
| 5 | [min()](http://www.tutorialspoint.com/java/number_min.htm) Returns the smaller of the two arguments. |
| 6 | [max()](http://www.tutorialspoint.com/java/number_max.htm) Returns the larger of the two arguments. |
| 7 | [cbrt()](http://www.tutorialspoint.com/java/number_log.htm) Returns the cube root value of the argument. |
| 8 | [pow()](http://www.tutorialspoint.com/java/number_pow.htm) Returns the value of the first argument raised to the power of the second argument. |
| 9 | [sqrt()](http://www.tutorialspoint.com/java/number_sqrt.htm) Returns the square root of the argument. |
| 10 | [round()](http://www.tutorialspoint.com/java/number_sin.htm) Returns the closest long or int value. |
| 11 | [random()](http://www.tutorialspoint.com/java/number_random.htm) Returns a random number between 0 and 1 |

Syntax: double b= Math.max(5,8); output : 8

double x =Math. cbrt(5,2); output:25

**LIBRARY FUNCTIONS**

**String Class**

Strings, which are widely used in Java programming, are a sequence of characters. InJava programming language, strings are objects.

The Java platform provides the String class to create and manipulate strings.

Immutable classes are Java classes whose objects cannot be modified once created. Any modification in Immutable objects result in new object.

String is immutable in Java.

Creating Strings:

The most direct way to create a string is to write:

**String greeting = "Hello world!";**

Whenever it encounters a string literal in your code, the compiler creates a String object with its value in this case, "Hello world!'.

You can create String objects by using the new keyword and a constructor.

**public class StringDemo{**

**public void main(){**

**String str =”hello”;**

**String helloString =new String(str);**

**System.out.println(helloString);**

**}**

**}**

This would produce result:

Hello

**Strings are immutable in nature.**

String is a class object, can hold unchanging string.

i.e., once initialized its content cannot be changed .

String class is defined in the java.lang.package

We can create string objects by using :

* String literal (implicitly)
* Using String constructor (Explicitly)

The String object can be created explicitly by using the new keyword and a constructor in the same way as you have created objects previously.

String str1 = "Java is Hot"; // Implicit construction via string literal

String str2 = new String("I'm cool"); // Explicit construction via new

Here is the list methods supported by String class:

**SN Methods with Description**

1 **length()**

To find the length(number of characters) of the string

Return type int

Argument nil

Format int len=”India”.length();

Output 5

2 **charAt()**

To extract one character at a time from the string

Return type char

Argument int

Format char ch.=”india”.charAt(2);

Output d

3 **compareTo()**

To compare two strings

Return type int

Argument String

Format System.out.print(”india”.compareTo(“India”)>0);

Output 32

If(string1.compareTo(string2)==0)output equal 0

If(string1.compareTo(string2)>0) output number>0

If(string1.compareTo(string2)<0) output number<0

4 **compareToIgnoreCase()**

To compare two strings

Return type int

Argument String

Format System.out.print(”india”.compareTo(“India”)>0);

Output 32

Format1:

System.out.print("ink".compareToIgnoreCase("INT"));

Output: -9

Format2:

System.out.print(“Ink”.compareToIgnoreCase(“INK”));

Output: 0

System.out.print(“Ink”.compareToIgnoreCase(“INKS”));

Output: -1

System.out.print(“Integer”.compareToIgnoreCase(“INT”));

Output: 4

5 **replace()**

To replace old character with new character

Return type String

Argument char,char

Format System.out.print(”india”.replace(‘i’, ‘a’));

Output andaa

6 **concat()**

To add two strings

Return type String

Argument String

Format System.out.print(”Great”.compareTo(“India”))

Output GreatIndia

7 **toLowerCase()**

To convert all the characters of the string into lowercase.

Return type String

Argument nil

Format System.out.print(”INDIA”toLowerCase());

Output india

8 **toUpperCase( )**

To convert all the characters of the string into uppercase.

Return type String

Argument nil

Format System.out.print(”india”.toUpperCase());

Output INDIA

9 **startsWith()**

To check whether the string is starting with the specified string

Return type boolean

Argument string

Format System.out.print(”india”.startsWith(“In”));

Output false

10 **endsWith()**

To check whether the string is ending with the specified string

Return type boolean

Argument string

Format System.out.print(”india”.endsWith(“in”));

Output false

11 **equals()**

To check whether the strings are having same number of characters and in th same sequence

Return type boolean

Argument string

Format System.out.print(”india”.equals(“indian”));

Output false

12 **equalsIgnoreCase()**

To ignore the case while checking their equality

Return type boolean

Argument string

Format System.out.print(”India”.equals(“india”));

Output true

13 **lastIndexOf()**

To tell the position of the last occurrence of the given character in the string

Return type int

Argument char

Format System.out.print(”India is great”.lastIndexOf(‘a’));

Output 12

.

14 **indexOf()**

To tell the position of the first occurrence of the given character in the string

Return type int

Argument char

Format 1 System.out.print(”India is great”.indexOf(‘a’));

Output 4

Format 2 System.out.print(”India is great”.indexOf(‘i’ , 4));

Output 6

If character is not found in the string -1 will be displayed in both 13th and 14th point.

15 **substring()**

To get the part of the string from the main string.

Return type String

Argument int

Format 1 System.out.print(”India is great”.substring(6));

Output is great (from that particular position till end)

Format 2 System.out.print(”We are Indians”.substring(1,8));

Output e are I (will start from 1st position till 7th position without including last character)

16 **trim()**

To remove blank spaces(white spaces) from the beginning and end of the string.

Return type String

Argument nil

Format System.out.print(” India is great ”.trim());

Output India is great (without blank spaces)

17  **valueOf()**

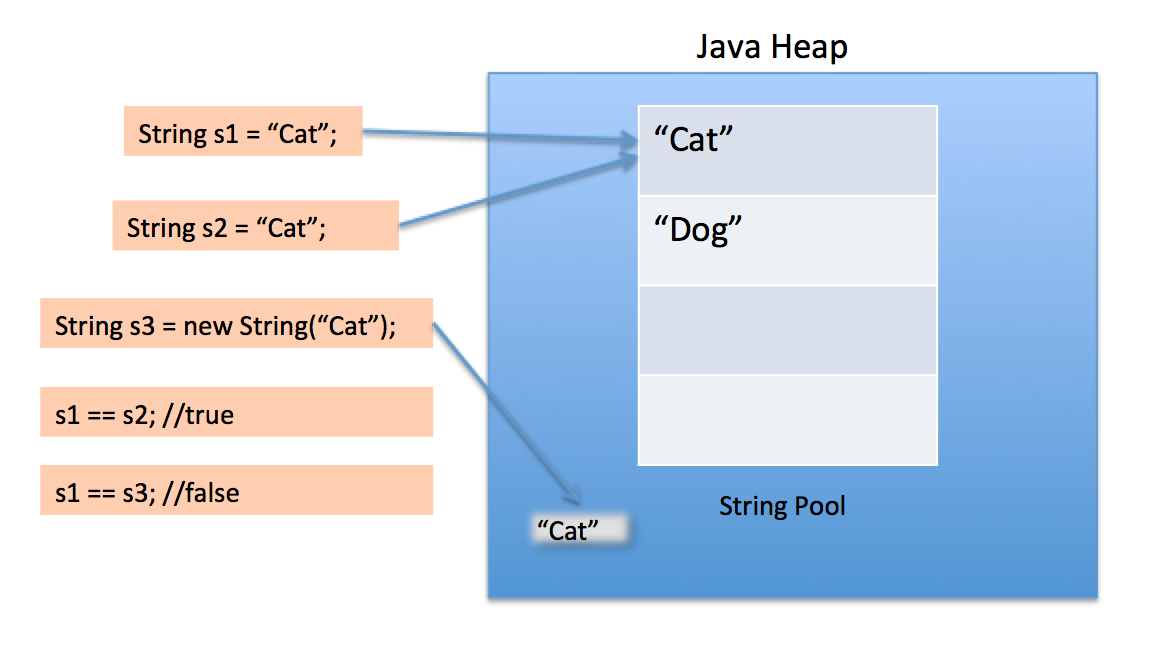
To convert string into number of any primitive data type

Return type int/float/long etc

Argument String

Format System.out.print(”678”.valueOf());

Output 678(can be used to do any type of calculation)



classstringg

{voidmain()

{Strings1="cat";

Strings2="cat";

Strings3=newString("cat");

if( s1.equals(s2))

System.out.println(s1);

else

System.out.println("no");

if( s2.equals(s3))

System.out.println(s3);

else

System.out.println("no");

}

}

Output

cat

no

|  |  |
| --- | --- |
| compareTo() | equals () |
| It compares two string not only for equality but for bigger, smaller than the other. | It compares strings only for equality. |
| It returns integer value | It returnboolean value. |

**LIBRARY FUNCTIONS**

**Character class**

Here is the list of the important character methods

|  |  |
| --- | --- |
| **SN** | **Methods with Description** |
| 1 | [isLetter()](http://www.tutorialspoint.com/java/character_isletter.htm) Determines whether the specified char value is a letter. |
| 2 | [isDigit()](http://www.tutorialspoint.com/java/character_isdigit.htm) Determines whether the specified char value is a digit. |
| 3 | [isWhitespace()](http://www.tutorialspoint.com/java/character_iswhitespace.htm) Determines whether the specified char value is white space.( Tab, Space and New line) |
| 4 | [isUpperCase()](http://www.tutorialspoint.com/java/character_isuppercase.htm) Determines whether the specified char value is uppercase. |
| 5 | [isLowerCase()](http://www.tutorialspoint.com/java/character_islowercase.htm) Determines whether the specified char value is lowercase. |
| 6 | [isLetterOrDigit()](http://www.tutorialspoint.com/java/character_isletter.htm) Determines whether the specified char value is a letter or digit. |
| 7 | [toLowerCase()](http://www.tutorialspoint.com/java/character_tolowercase.htm) Returns the lowercase form of the specified char value. |
| 8 | [toUpperCase()](http://www.tutorialspoint.com/java/character_touppercase.htm) Returns the uppercase form of the specified char value. |

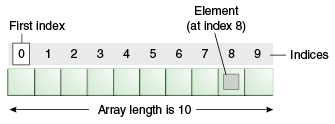
Syntax: Boolean b= Character.isUpperCase/isLetter(ch)

char x =Character. toUpperCase/toLowerCase(ch)

Where ch is char type.

**Arrays**

An *array* is a container that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.

An array of 10 elements.

Each item in an array is called an *element*, and each element is accessed by its numerical *index*. As shown in the preceding illustration, numbering begins with 0. The 9th element, for example, would therefore be accessed at index 8.

The following program, [ArrayDemo](http://docs.oracle.com/javase/tutorial/java/nutsandbolts/examples/ArrayDemo.java), creates an array of integers, puts some values in the array, and prints each value to standard output.

class ArrayDemo {

public void main() {

// declares an array of integers

int[] num;

int num[ ];

// allocates memory for 10 integers

num = new int[10];

**Declaring a Variable to Refer to an Array**

The preceding program declares an array (named anArray) with the following line of code:

// declares an array of integers

int[] anArray;

Or

int anArray[ ];

Like declarations for variables of other types, an array declaration has two components:

* + - * the array's type
      * the array's name.

-An array's type is written as *type*[], where *type* is the data type of the contained elements; the brackets are special symbols indicating that this variable holds an array.

-The size of the array is not part of its type (which is why the brackets are empty).

-An array's name can be anything you want, provided that it follows the rules.

-The declaration does not actually create an array; it simply tells the compiler that this variable will hold an array of the specified type.

Similarly, you can declare arrays of other types:

byte[] anArrayOfBytes;

short[] anArrayOfShorts;

long[] anArrayOfLongs;

float[] anArrayOfFloats;

double[] anArrayOfDoubles;

boolean[] anArrayOfBooleans;

char[] anArrayOfChars;

String[] anArrayOfStrings;

You can also place the brackets after the array's name:

float anArrayOfFloats[];

booleananArrayOfBooleans[ ];

char anArrayOfChars[ ];

String anArrayOfStrings[ ];

**Creating, Initializing, and Accessing an Array**

One way to create an array is with the new operator. The next statement in the ArrayDemo program allocates an array with enough memory for 10 integer elements and assigns the array to the anArray variable.

// create an array of integers

anArray = new int[10];

The next few lines assign values to each element of the array:

anArray[0] = 100; // initialize first element

anArray[1] = 200; // initialize second element

anArray[2] = 300; // and so forth

Each array element is accessed by its numerical index:

System.out.println("Element 1 at index 0: " + anArray[0]);

System.out.println("Element 2 at index 1: " + anArray[1]);

System.out.println("Element 3 at index 2: " + anArray[2]);

Alternatively, you can use the shortcut syntax to create and initialize an array:

int[] anArray = { 100, 200, 300,400, 500, 600,700, 800, 900, 1000};

**LENGTH of an ARRAY**

Length is a keyword to calculate the length of an array

Here the length of the array is determined by the number of values provided between braces and separated by commas.

char anArray[ ]={‘w’,’o’,’p’,’q’,a’’}; // initialization

int len=anArray.length; // length of anArray is 5.

**ARRAY as PARAMETER**

Array can be passed to method just like other primitive type

class value

{ void main (int arr[ ])

{int i;

for(i=0;i<arr.length;i++)

{ System.out.println(arr[i];) }

}}

**Arrays are dynamic in nature:**

It stores value at the time of execution

**ARRAY VARIABLE or SUBSCRIPTED VARIABLE:**

A user defined name to identify a set of adjacent memory locations is called array variables or a subscripted variable.

E.g. int a[ ] =new int[10]; Here a is an array variable and 10 is the size of an array.

**SUBSCRIPTED or INDEX :**

An array variable is accompanied by a subscript to identify a specific memory location. A subscript is actually an integer.

Here, size 10, subscript of array variable from 0 to 9.

* num[9] is the 10th element of the array num.
* The first index of the array is called LOWER BOUND.
* The last index of the array is called UPPER BOUND.
* When an array object is created with the new operator,Its elements are automatically initialized to zero, which is the default initial value for all numeric types.
* Java array’s index numbering starts with 0 zero.
* The data type of array elements is known as the base type of the array.
* The element numbers in [ ] are called subscripts or indices.
* The subscripts other than 0 to n-1( both inclusive) for an array having n elements, are called **out-of-bounds subscripts**

## SEARCHING

## Linear and Binary Search

**Linear Search and Binary Search.** Both the techniques are different and are  used in different situations.

## Linear Search:

1. This method is used when the volume of data is small.
2. It works for sorted and unsorted data set both.
3. In this method all the data elements are scanned in a linear fashion one by one till the element to be searched is found or no more data elements  are left in the list.

## **Binary Search:**

1. It is good for large volume of data.
2. The pre-condition for binary search is that the array should be sorted in some order.
3. In this method, the array is divided in two haves and the mid value is found. The element to be searched is matched with the mid value. If it matches, the user is told about it and loop breaks. If it does not match then it is checked whether the value to be searched is greater than the mid-value or lesser. If it is greater, then the lower half of array is ignored and the upper half of array is divided to find the mid-value and the process repeats. (It works because the array is sorted). Same process repeats if the value to be searched is in the lower half of the array.

|  |  |
| --- | --- |
| Linear Search | Binary Serach |
| Linear search works on sorted and unsorted data items. | Binary search takes place only on sorted data items ie either in ascending or descending order. |
| The search begins at the start of an array.ie 0th position | An array is divided into halves and then the desired data item is searched either in the first half or in the second half. |

**LINEAR SEARCH**

**Sample program:**

**/\*\* to enter numbers in an array and print the message found if**

**users number is in an array\*/**

**public class linear\_search**

**{**

**void main(int n[],int nn)**

**{int i,flag =1;**

**for(i=0;i<n.length;i++)**

**{if(n[i]==nn)**

**flag=10;**

**}**

**if(flag==10)**

**System.out.print("found");**

**else**

**System.out.print("not found");**

**}**

**}**

**BINARY SEARCH**

**Sample program:**

**public class binary\_search**

**{/\*\* Search user's number in an array \*/**

**void main(int a[],int n)**

**{int m=0,b=0,e= a.length-1,p=-1;**

**while(b<=e && p==-1)**

**{m=(b+e)/2;**

**if(a[m]>n)**

**e=m-1;**

**if(a[m]<n)**

**b=m+1;**

**if(a[m]==n)**

**p=m;**

**}**

**if(p==-1)**

**System.out.print("not found");**

**else**

**System.out.print(p);**

**}**

**}**

**SORTING**

**BUBBLE and SELECTION**

Sorting of an array means arranging the elements in a specified order i.e. either ascending or descending order. There are several technique available.

* Bubble sort
* Selection sort

**Bubble Sorting**

The basic idea of bubble sort is to move the largest element to the highest index position in the array. To attain this, two adjacent elements are compared and exchanged if they are not in proper order.

Sample program:

public class bubble\_sort

{/\*\* arrays of string and numeric type arranging in ascending order on the basis of

string array\*/

void m(String n[], int a[])

{ for(int i=0;i<n.length;i++)

{for(int j=0;j<n.length-1;j++)

{if(n[j].compareTo(n[j+1])>0)

{int t=a[j];

a[j]=a[j+1];

a[j+1]=t;

String tt=n[j];

n[j]=n[j+1];

n[j+1]=tt;

}}}

for(int i=0;i<n.length;i++)

System.out.println(n[i]+"\t"+a[i]);

}

}

**Selection Sort:** In selection sort, the smallest value from the remaining unsorted array is searched for and put in the sorted array. This process repeats until the entire array is sorted.

**public class Selection\_sort**

**{/\*\* array of numeric type arranged in descending order \*/**

**void m(int a[])**

**{ int s=0,l=0;**

**for(int i=0;i<a.length;i++)**

**{s=a[i]; l=0;**

**for(int j=i+1;j<a.length-1;j++)**

**{**

**if(s>a[j])**

**{s=a[j];**

**l=j;**

**}**

**int t=a[l];**

**a[l]=a[i];**

**a[i]=t;**

**}**

**}**

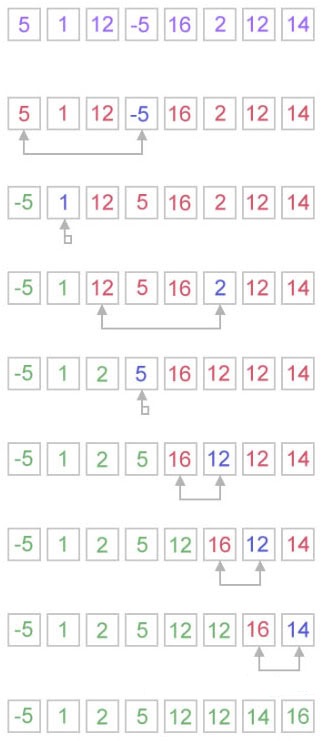
**for(int i=0;i<a.length;i++)**

**System.out.println(a[i]);**

**}**

**}**

Example. Sort {5, 1, 12, -5, 16, 2, 12, 14} using selection sort



**Difference between**

**Call by value  
Call by reference**

There are two ways of invoking a method

in Java

* Call By Value
* Call By Reference
  1. **Call by Value:**Call by Value is the method of invoking a method by passing a copy of the actual parameters to the formal parameters such that any change occurring in the formal parameters will not reflect on the actual parameters.

Consider the following example:

**class** Sample

{**void**accept(**int** aa, **int** bb)

{ i**nt** sum = aa + bb;

System.**out**.println(sum); }

**void** main ()

{

**int** a = 6;

**int** b = 10;

accept(a,b); }

**Call by Value/Pass by Value**

* In call by value actual and formal parameters are variables (primitive data types)
* In call by value actual parameters are copied their value to formal parameters.
* In call by value, actual and formal parameters both have different memory locations.
* If any changes done to the formal parameters, they are not reflected back in the actual parameters.
  1. **Call by Reference:** By this method, the actual reference or actual location of the actual parameters are passed to the formal parameters such that any changes occurring in the formal parameters will reflect on the actual parameters.

**class** Sample1

{ **void** Add(**int** x[]) {

**for** (int i = 0; i <x.length; i++)

{System.**out**.**print**(x[i]\*2 + " ");

System.**out**.println(); } }

**void** main ()

{ **int** a[] = {3, 6, 8, 9};

Add(a);

System.**out**.println("The arguments after function call: ");

**for** (int j = 0; j <a.length; j++)

{System.**out**.println(a[j] + " ");

System.**out**.println(); }}

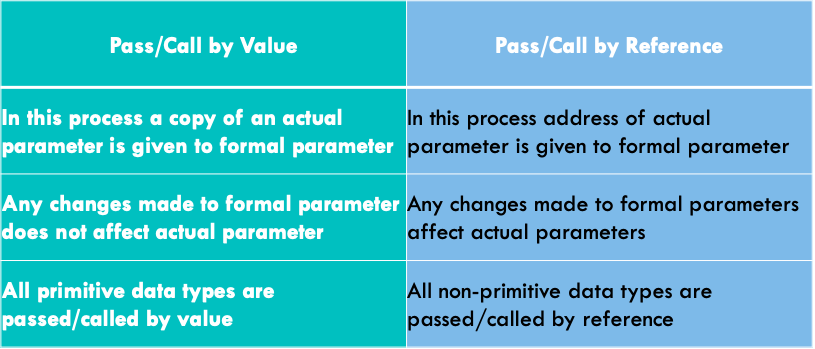
**Call by Reference/ Pass by Reference**

1. In call by reference actual and formal parameters are objects (Reference data types)

2. In call by reference actual parameters are copied their reference number to formal parameters.

3. In call by reference, actual and formal parameters both have same memory locations.

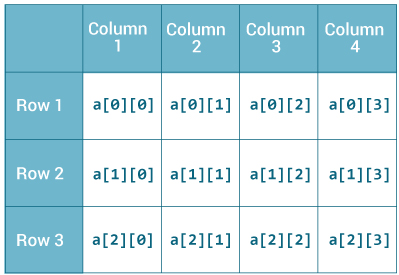
4. If any changes done to the formal parameters, they are reflected back in the actual parameters because of same memory location.



**TWO DIMENSIONAL ARRAY**

A two-dimensional array is an array of one dimensional arrays

* A double dimensional array is that in which even the columns along with the rows can be declared
* *Example ,*int m[2][4] = {{1,5,7,6},{3,8,9,6}};
* Two dimensional(2D) arrays are indexed by two subscripts, one for the row and one for the column.
* Example:  
  *row col*
* In this chapter, you learned to create and use array of primitive data types (like: Double, int etc.) and String array
* int [][] a = new int[3][4];
* Here, a is a two-dimensional (2d) array. The array can hold maximum of 12 elements of type int.



Remember, Java uses zero-based indexing, that is, indexing of arrays in Java starts with 0 and not 1.

**Declaring 2D Arrays**

* Declare a local variable **rating** that references a 2D array of int:

**int [][] rating;**

* •Create a 2Darray with 3 rows and 4 columns and assign the reference to the new array to rating:

**rating = new int[3][4];**

* Short cut to declare and create a 2D array:

**int [][] rating = new int[3][4];**

**Size of 2D Arrays**

* Given  
  **int [][] rating = new int[3][4];**

number of rows (first dimension) =3

number of columns (second dimension) =4

* **Example 2**

Find the number of ratings above the value of the parameter.

**public int countAbove(int [][] rating) {**

**for (int row = 0; row <rating.length ; row++)**

**{**

**for (int col = 0; col <rating[0].length; col++)**

**{**

**How to initialize a 2d array in Java?**

Here's an example to initialize a 2d array in Java.

int [][] a = { {1, 2, 3},

{4, 5, 6},

{7, 8, 9} };

OR

int [][] a = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };

**Ways To Input the elements of the Arrays**

**1. By Assignment Method**

int m[][]={{1,5,6,9},{2,5,9,7},{15,5,78,9}};

**2. By Method Parameter**

void compute(int m[][])

**3. By Input Stream Reader Method( using scanner)**

**m[i][k]=sc.nextInt();**

**Similarity with 1D Arrays**

* Each element in the 2D array must be of the same type, either a primitive type or object type.
* Subscripted variables can be used just like a variable:

**rating[0][3] = 10;**

* Array indices must be of type int and can be a literal, variable, or expression.

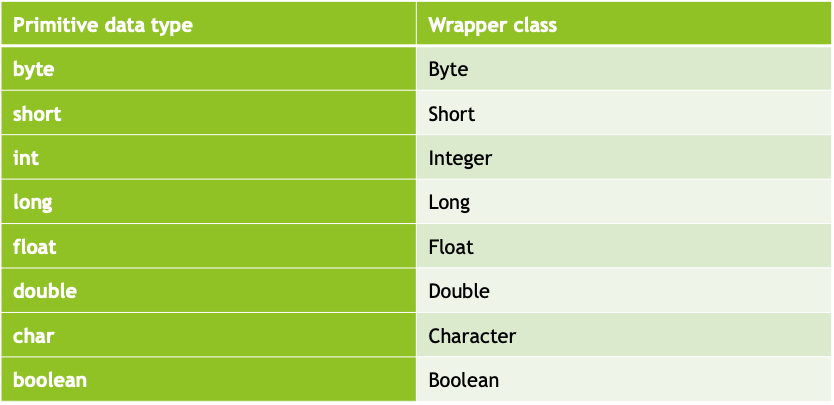
**rating[3][j] = j;**

* If an array element does not exists, the Java runtime system will give you an

**ArrayIndexOutOfBoundsException**

**WRAPPER CLASS**

* As the name says, a wrapper class wraps (encloses) around a data type and gives it an object appearance. Wherever, the data type is required as an object, this object can be used.
* Wrapper classes include methods to un wrap the object and give back the data type.
* It can be compared with a chocolate. The manufacturer wraps the chocolate with some foil or paper to prevent from pollution. The user takes the chocolate, removes and throws the wrapper and eats it.
* Wrapper classes are part of **java.lang**package.
* Wrapper classes are those classes which exists for every primitive data type.
* They are used to convert any data type into an object.
* The primitive data types are not objects; they do not belong to any class. Sometimes, it is required to convert data types into objects in Java language



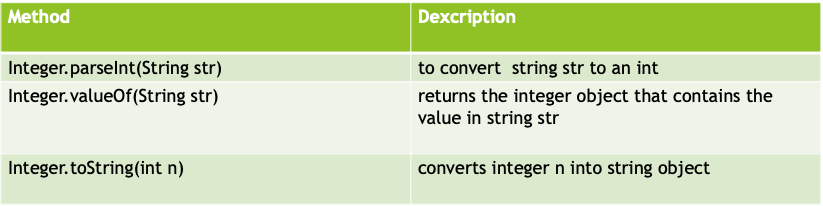
**Why we need wrapper class**

A wrapper class provides many methods for conversion between primitive data types and String. Some commonly used methods are:

Syntax: int t= Integer.parseInt(String)

Integer t= Integer.valueOf(String)

String dd=Double.toString(10.0);



Sample program:

String s1=“36”,s2=“64”;

{double cc=Double.parseDouble(s1)+Double.parseDouble(s2);

Integer m= Integer.valueOf(s1);

System.out.print(Math.sqrt(cc)+“\n“+ m); }

Output: 10

36