**Java Operators**

Java operators fall into different categories:

**Arithmetic Operator**

public class Test {

public static void main(String args[]) {

System.out.println("Integer Arithmetic");

int i = 1 + 1;

int n = i \* 3;

int m = n / 4;

int p = m - i;

int q = -p;

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

System.out.println("n = " + n);

System.out.println("m = " + m);

System.out.println("p = " + p);

System.out.println("q = " + q);

// arithmetic using doubles

System.out.println("\nFloating Point Arithmetic");

double a = 1 + 1;

double b = a \* 3;

double c = b / 4;

double d = c - a;

double e = -d;

System.out.println("a = " + a);

System.out.println("b = " + b);

System.out.println("c = " + c);

System.out.println("d = " + d);

System.out.println("e = " + e);

}

}

Integer Arithmetic

i = 2

n = 6

m = 1

p = -1

q = 1

Floating Point Arithmetic

a = 2.0

b = 6.0

c = 1.5

d = -0.5

e = 0.5

**Relational Operator**

public class Test {

public static void main(String args[]) {

int a = 10;

int b = 20;

System.out.println("a == b = " + (a == b));

System.out.println("a != b = " + (a != b));

System.out.println("a > b = " + (a > b));

System.out.println("a < b = " + (a < b));

System.out.println("b >= a = " + (b >= a));

System.out.println("b <= a = " + (b <= a));

}

}

Output:

a == b = false

a != b = true

a > b = false

a < b = true

b >= a = true

b <= a = false

**Logical Operator**

public class Test {

public static void main(String args[]) {

boolean a = true;

boolean b = false;

System.out.println("a && b = " + (a && b));

System.out.println("a || b = " + (a || b));

System.out.println("!(a && b) = " + !(a && b));

}

}

Output:

a && b = false

a || b = true

!(a && b) = true

**Bitwise operators**

**Note:**

“>>>” is unsigned bit shift “>>” and “<<” are signed bit shift

**Conditional Operator:**

if (a > b) {

max = a;

}

else {

max = b;

}

Setting a single variable to one of two states based on single condition is such a common use of if-else that a shortcut has been devised for it, here is it.

max = (a > b) ? a : b;

**Precedence Rule**

Java has well-defined rules for specifying the order in which the operators in an expression are evaluated when the expression has several operators.

public class Precedence {

public static void main(String[] args) {

System.out.println(3 + 3 \* 2);

System.out.println(3 \* 3 - 2);

System.out.println(3 \* 3 / 2);

System.out.println("--");

System.out.println(1 \* 1 + 1 \* 1);

System.out.println(1 + 1 / 1 - 1);

System.out.println(3 \* 3 / 2 + 2);

}

}

Output:

9

7

4

--

2

1

6

**Scope of Variables**

The scope of a variable defines the section of the code in which the variable is visible. As a general rule, variables that are defined within a block are not accessible outside that block. The lifetime of a variable refers to how long the variable exists before it is destroyed.

These example will give your better understanding of scope of variable.

public class Test {

public static void main(String args[]) {

{

int x = 10;

System.out.println(x);

}

System.out.println(x); // error x not visible here

}

}

class Test {

public static void main(String args[]) {

{

int x = 5;

{

int x = 1010; // error duplicate local variable x System.out.println(x);

}

}

}

}

class Test {

public static void main(String args[])

{

for (int x=0; x<4; x++)

{

System.out.println(x);

}

System.out.println(x) //error

// x cannot be resolved to a variable

}

}

**Method:**

**What a method(or function) is ?**

A function is a block of statements, which is used to perform a specific task. Suppose you are building an application in Java language and in one of your program, you need to perform a same task more than once. So in such scenario you have two options –

1. Use the same set of statements every time you want to perform the task
2. Create a function, which would do the task, and just call it every time you need to perform the same task.

**Why we need a method?**

Functions are used because of following reasons –

1. To improve the readability of code.
2. Improves the reusability of the code, same function can be used in any program rather than writing the same code from scratch.
3. Debugging of the code would be easier if you use functions, as errors are easy to be traced.
4. Reduces the size of the code, duplicate set of statements are replaced by function calls.

**Syntax of defining a method**

modifier returnType nameOfMethod (Parameter List) {

// method body

}

The syntax shown above includes:

* modifier: will study this later
* returnType: method may return a value.
* nameOfMethod: it is advised to have a meaningful name for the method so that it would be easy to understand the purpose of method just by seeing it’s name.
* Parameter List: it is the type, order, and number of parameters of a method, these are optional, method may contain zero parameters.
* method body: method body defines what the method does with statements.

**Method Calling**

For using a method, it should be called. When a program invokes a method, the program control gets transferred to the called method. There are two ways in which a method is called:

1. when a method returns a value
2. when a method returns no value

**when a method returns value:**

public class Test {

public static void main(String[] args) {

int a = 11;

int b = 6;

int c = minFunction(a, b);

System.out.println("Minimum Value = " + c);

}

/\*\* returns the minimum of two numbers \*/

public static int minFunction(int n1, int n2) {

int min;

if (n1 > n2)

min = n2;

else

min = n1;

return min;

}

}

**when a method returns no value:**

The void keyword allows us to create methods which do not return a value. Here, in the following example we're considering method fun() that is a void method which does not return any value.

public class ExampleVoid {

public static void main(String[] args) {

fun(255.7);

}

public static void fun(double points) {

if (points >= 202.5) {

System.out.println("Rank:A1");

}

else if (points >= 122.4) {

System.out.println("Rank:A2");

}

else {

System.out.println("Rank:A3");

}

}

}

Output:

Rank:A1

**What is call by value ?**

The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the method have no effect on the argument. Java passes method arguments by value

public class CallByValue {

public static void main ( String[] args ) {

int x =3;

System.out.println ( "Value of x before calling increment() is "+x);

increment(x);

System.out.println ( "Value of x after calling increment() is "+x);

}

public static void increment ( int a ) {

System.out.println ( "Value of a before incrementing is "+a);

a= a+1;

System.out.println ( "Value of a after incrementing is "+a);

}

}

Output:

Value of x before calling increment() is 3

Value of a before incrementing is 3

Value of a after incrementing is 4

Value of x after calling increment() is 3

**Note:**

There are no default parameters in java.

Example :

void function(int num = 10){} //not allowed

**Array**

An Array is a collection of variables of the same type.

For instance an array of int is a collection of variables of type int. The variables in the array are ordered and each have an index.

**Initializing Arrays:**

int [] temps = new int [3];

temps[0] = 78; //filling one element at a time

temps[1] = 88;

temps[2] = 53;

**Initialize array to take user input:**

Scanner s= new Scanner(System.in);

int [] nums = new int [8];

for(int ctr = 0; ctr < 8; ctr++) {

nums[ctr] = s.nextInt();

}

**Initialize at time of declaration:**

double [ ] temperature = { 13.5, 18.4, 19.6, 21.4};

**Finding max value from array:**

public class TestArray {

public static void main(String[] args) {

double[] myList = {1.9, 2.9, 3.4, 3.5};

// Print all the array elements

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

System.out.println(myList[i] + " ");

}

// Summing all elements

double total = 0;

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

total += myList[i];

}

System.out.println("Total is " + total);

// Finding the largest element

double max = myList[0];

for (int i = 1; i < myList.length; i++) {

if (myList[i] > max) {

max = myList[i];

}

}

System.out.println("Max is " + max);

}

}

**Returning array from method**

class ArrayUse{

public static void main(String[] args){

int[] A = numbers();

}

public static int[] numbers(){

int[] A = new int[3];

A[0] = 2;

A[1] = 3;

A[2] = 4;

return A;

}

}

**Passing array to method:**

class Testarray {

static void min(int arr[]) {

int min = arr[0];

for (int i = 1; i < arr.length; i++) {

if (min > arr[i]) {

min = arr[i];

}

System.out.println(min);

}

public static void main(String args[]) {

int a[] = { 33, 3, 4, 5 };

min(a);

}

}

**Reverse an array**

static void reverse(int arr[]) {

int temp;

for (int i = 0; i < arr.length/2; i++) {

temp = arr[i];

arr[i] = arr[arr.length-1-i];

arr[arr.length-1-i] = temp;

}

}

public static void main(String[] args){

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

reverse(a);

for(int i:a){

System.out.print(i+" ");

}

}

Output:

6 5 4 3 2 1

**Bubble Sort**

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

public static void BubbleSort(int array[]){

int temp = 0;

for(int i=0;i<array.length-1;i++){

int flag =1;

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

if(array[j]>array[j+1]){

temp = array[j];

array[j] = array[j+1];

array[j+1] = temp;

flag =0;

}

}

if(flag == 1)

break;

}

}

**Selection Sort**

The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

public static void sort(int array[]){

int smallest = -1,temp=0;

for(int i=0;i<array.length-1;i++){

smallest = i;

for(int j=i+1;j<array.length;j++){

if(array[smallest]>array[j]){

smallest = j;

}

}

temp = array[smallest];

array[smallest] = array[i];

array[i] = temp;

}

}

**Insertion Sort**

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, insertion sort.

The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array).

public static void sort(int array[]){

int temp = 0,i=0,j=0;

for(i=1;i<array.length;i++){

temp = array[i];

for(j=i-1;j>=0 && temp<array[j];j--){

array[j+1]=array[j];

}

array[j+1] = temp;

}

}