Core JAVA

INTRODUCTION

- ***Introduction***
- ***Java Buzzwords***
- ***Installing JDK Software***
- ***class & object***
- ***keywords***
- ***Identifiers***
- ***Datatypes***
- ***Arrays***
- ***main method***
- ***command line arguments***

Introduction

The early **1950** languages such as **Fortran** and **Cobol** are Specific purpose languages. The main drawback with specific purpose languages is they are specific to a particular purpose. For example, Fortran is meant for developing only scientific applications, and Cobol is meant for developing only business oriented applications.

In early **1960s**, the **BCPL** (Basic Combined Programming Language) language was developed for developing all types of applications i.e., BCPL is a general purpose language. The drawbacks in BCPL are solved with a new language called '**B**'. Even though 'B' is a general purpose language, it has its own drawbacks. To solve these problems, a powerful, efficient, reliable language **"C"** was developed by Dennis Ritche in 1972.

The 'C' language follows Structured Programming approach.

The main drawback with Structured Programming languages such as 'C' is its complexity increases once the project size reaches a certain size.

To solve this problem, a new way to program was invented called Object Oriented Programming (OOP).

OOP programming methods helps us to organize (simplify) complex programs through Encapsulation, Inheritance, and Polymorphism.

C++ was developed by Stroustrup in 1979.

Two main drawbacks with C++ language are : Platform dependent and won't support Internet and WWW.

To solve these problems, the **James Gosling** developed a new language called **JAVA** at sun Microsystems in **1991**, which is platform independent and supports WWW & Internet.

Java was initially called as "OAK" after he saw oak tree outside of his window at sun. Later he came to know that the Language "OAK" already existed in the market. When a group of sun people visited a local coffee shop, the name "JAVA" was suggested. Java is a slag term for coffee.

Java Buzzwords

The following are Java buzzwords:

1. Simple
2. Object Oriented

3. Robust
4. Platform Independent
5. Portable
6. Multithreaded (Or Parallel Execution)
7. Distributed (Or Networking)
8. Secure
9. Dynamic

I. Simple

Java is a simple language because of the following reasons:

- Java is syntactically related to C++ which is a direct decedent from C (i.e., C++ is evolved from C). Most of the syntax and characters are inherited from these two languages.
- Some of the most confusing concepts in C/C++ are either left out of java or implemented in a cleaner way.
  For example: Pointers are completely eliminated from java. The disadvantage with pointers are overriding memory and corrupting data. In java, the memory management is implemented in a cleaner way using Garbage Collector (GC)

II. Object Oriented

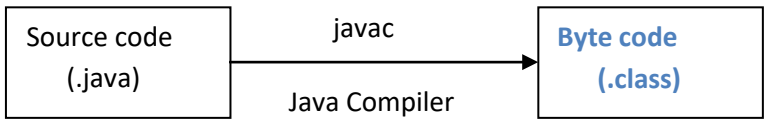
Java language follows Object Oriented Programming (OOP) paradigm.
The OOP principles are Encapsulation, Inheritance, and Polymorphism.
Every Program in java is an Object Oriented Program. Java is a fully object oriented language but not pure object oriented language because of the existence of primitive data types. Small talk is a pure object oriented language. Literally everything in smalltalk is an object.

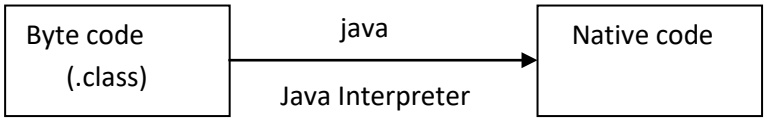
III. Robust (Means Reliable)

Java is intended for writing reliable programs. Java provides Exception Handling mechanism which puts lots of emphasis on early checking for possible errors and dynamic checking. Java automatically handles Memory Management using Garbage collection. Java eliminates pointers.

IV. **Platform Independent**

Write once, Run on any platform, Any time, Forever. Java is a platform independent language because of the **bytecode**. The compiled bytecode is **common** across all operating systems (platforms).
The Java source files are compiled to a bytecode, where as C/C++ source files are directly compiled to a native code.





V. Portable

The size of the primitive data types are fixed on all platforms.

Also, binary data is stored in a fixed format, eliminating the “big endian | Little Endian” confusion.

VI. Parallel Execution (or Multithreading)

Reduces the execution time I,e., improves execution speed.

VII. Networking (Or Distributed)

Java has become one of the most popular language for N/W programming due to two salient features: Platform independent & Multithreading.

VIII. Secure

Java provides several layers of protection from dangerous code, virus, Trojan horses. Java security is provided from the Java Architecture itself. Also, Java security can be provided using security API.

IX. Dynamic

It was designed to adapt to an evolving environment.

Installaling JDK Softwares

Step#1: Download latest stable Java software from SUN website (http://java.sun.com).
Step #2: Install JDK software by double click on .exe file.
Step #3: The following folders are created in JAVA_HOME after installing JDK.

%JAVA_HOME%/bin
%JAVA_HOME%/demo
%JAVA_HOME%/include
%JAVA_HOME%/jre
%JAVA_HOME%/lib

Step #4: Set PATH, CLASSPATH, JAVA_HOME
Once after installing java software, we have to set JAVA_HOME, PATH, and CLASSPATH environment variables. Though setting user defined environment variable JAVA_HOME is optional, but it is recommended to set.

Environment	Description	Remark
JAVA_HOME	Should set to java home directory where java is installed.	It is a user defined environment variable.
PATH	Should set to Java Resources folder (i.e., bin)	It is a system environment
CLASSPATH	Should set to java class files(%JAVA_HOME%/jre/lib/rt.jar).	It is a system environment variable.

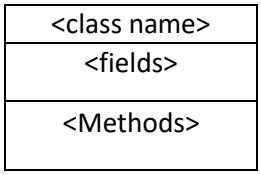
class and object

A class is a template or blueprint.

Syntax:

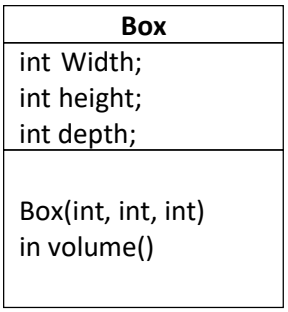
<class modifiers> **class** <class name> [extends clause] [implements clause]{
    [field declarations]
    [constructor declarations]
    [method declarations]
}

The **UML** (Unified Modeling Language) notation for the class is:



Example:

```
class Box{
        //Field declaration
        int  width;
        int height;
        int depth;
        //Constructor
        Box(int w, int h, int d){
                width = w;
                height = h;
                depth=d;
        }
        //Method declaration
        int volume(){
                return width*height*depth;
        }
}
```



object

Instance of a class is called as an object. In java, object is created using '**new'** keyword.
The object creation process involves following three steps:

- Declaring reference variable
- Construction an object using 'new' keyword.
- Assigning Reference value (address) to reference variable.

Example:

```
Box b = new Box(1,2,3);
```

The fields of a class is also called as attribute, properties, instance variable, or data member.
The methods of a class is also called as behavior, operation, instance method, or method member.
The data member and method member of a class is called as member of a class.
The **new** keyword creates an object and returns the reference value to the reference variable.

Keywords

The keyword has a **predefined meaning** in the programming language.

In jdk1.6, there are **50** keywords and **3** reserved literals.
The below table summaries keyword category for our better understanding:

Category	Keywords	No_Of_Keywords
Keywords for Primitive data type	byte, short, int, long, float, double, char, boolean, void	9
Keywords for Control flow	if, else, switch, case, default, while, do, for, break, continue, return	11
Keywords for Exception handling	try, catch, finally, throw, throws, assert	6
Keywords for Modifiers	private, protected, public, final, static, abstract, interface, strictfp, native, transient, volatile, synchronized	12
class related keywords	class, extends, implements, package, import, enum	6
Object related	new, this, super, instanceof	4
Unused keywords	const, goto	2

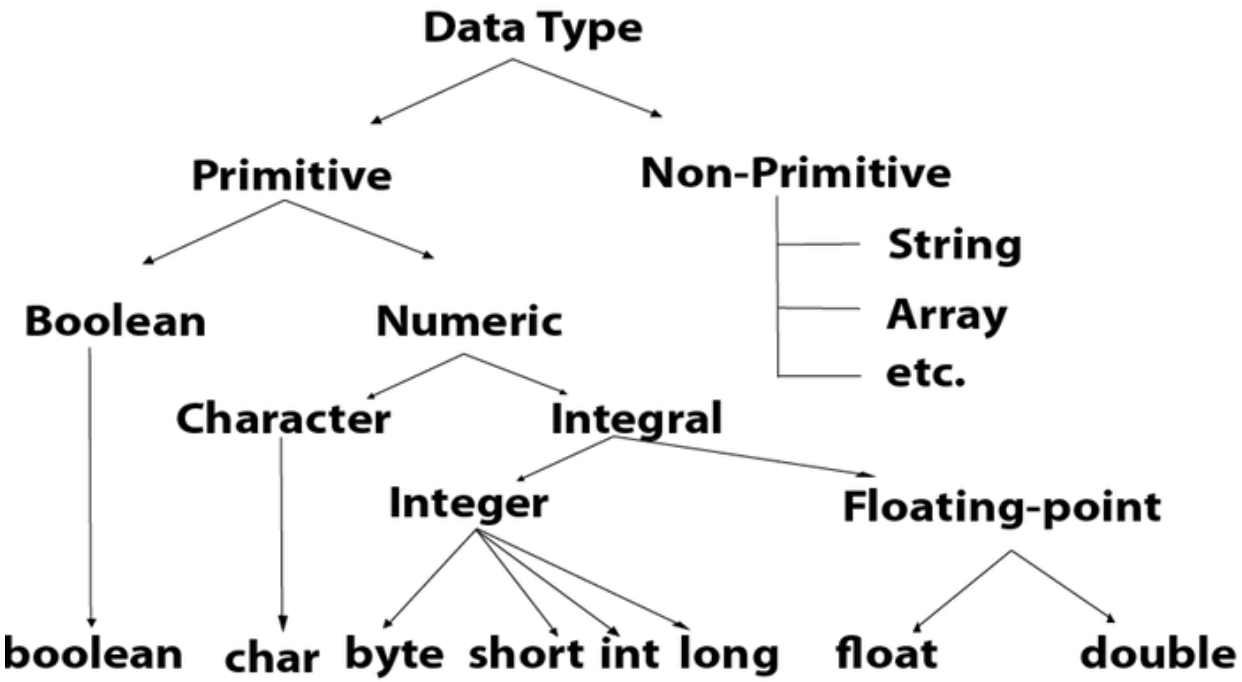
The new keyword added in jdk1.5 is: **enum.**
The new keyword added in jdk1.4 is: **assert**.
The new keyword added in jdk1.2 is: **strictfp**.

Identifiers

The **name** in a program is called an Identifier. Identifier is a sequence of characters. Identifiers are the names of the variables, methods, classes, packages, interfaces, arrays, enumerations, labeled names, etc.
Example:

```
class Welcome{
        public static void main(String[] args){
                int age = 10;
                System.out.println(“Welcome program”);
        }
}
```

Datatypes



In java, the data types are grouped into two categories:

- Primitive types
- Reference types → classes, interfaces, exceptions, errors, enums, annotations, and arrays

Primitive types

There are **9** primitive types in java.

Below table summarizes the categories of the primitive types:

Category	Primitive types
integers	byte, short, int, long
floating point numbers	float, double
Characters	char
Booleans	boolean
empty set data type	void

Below table summarizes the corresponding wrapper classes, size, range, and default values:

Primitive type	Wrapper type	Size (in bytes)	Range	Default values
byte	java.lang.Byte	1	-128 to +127	0
short	java.lang.Short	2	-32,768 to + 32,767	0
Int	java.lang.Integer	4	-2^{31} to $+2^{31}-1$	0
long	java.lang.Long	8	-2^{63} to $+2^{63}-1$	0
float	java.lang.Float	4	-3.4e38 to 3.4e38	0.0F
double	java.lang.Double	8	-1.7e308 to 1.7e308	0.0
Char	java.lang.Character	2	0 to 65,535	Blank Space(' ')
boolean	java.lang.Boolean	N/A	N/A	false
void	java.lang.Void	N/A	N/A	N/A

Note:

1. By default, all integers are initialized with 0.
2. By default, all floating point numbers are initialized with 0.0
3. By default, all characters are initialized with blank space.
4. By default, all booleans are initialized with false.
5. By default, all reference types are initialized with null.
6. The size of the primitive data types are **fixed** across all operating systems.
7. The java supports only signed integers but unsigned characters.

Arrays

- ***Introduction***
- ***Array Declaration***
- ***Array Construction***
- ***Array Initialization***
- ***Annonymous arrays***
- ***length vs length()***
- ***Disadvantages with arrays***

Introduction

An array is a group of **homogeneous** elements i.e., all elements in an array have same data type.

0	1	2	3	4	5	6	7	8	9

The array is an index based collection whose index starts at 0 and ends with index (**n-1**), **where 'n' is thenumber of elements in an array.**

No Of Elements(n) = 10;

End Value = n-1 (10-1) = 9 – Last Index or End Index

The size of an array is fixed and cannot be changed.

In java, arrays are created dynamically at runtime using **'new'** keyword, hence, arrays are objects.

The element types in an array are either Primitive or Object (reference) type.

Array Declaration

Single dimensional arrays are declared in following ways:

<array type> <array name>**[]**;

(or)

<array type>**[]** <array name>;

(or)

<array type> **[]**<array name>;

Example:

int arr1[], arr2[]; //Both arr1 and arr2 are arrays.

Int[] arr1, arr2;//Both arr1 and arr2 are arrays.
Int []arr1, arr2; //Only arr1 is an array but arr2 is not.

Array Construction

In java, arrays are dynamically created at runtime using **'new'** keyword, hence, arrays are objects.

Syntax:

<array type>[] <array-name> = **new <array type>[<array-size>];**

Example

int[] arr = **new int[10];**

Initializing Array Elements

a) Default Initialization

When an array is created using new keyword, all its elements are automatically **initialized to their default values.**

Example:

int[] arr = new int[3];

0	0	0

float[] arr = new float[3];

0.0F	0.0F	0.0F

boolean[] arr = new boolean[3];

false	False	False

String[] arr = new String[3];

Null	Null	null

b) Initialize array elements using Initialization block

Syntax:

<array type>[] <array name> = { <array initialization block> };

Example:

int[] age = {20,21,22,23};

c) Array Declaration, Construction, and Initilialization in a single step

Syntax:

<array type>[] <array name> = new <array type>**[]**{<array initialization block>};

Example:

int[] arr = new int[]{10,20,30}; //Ok.

d) Anonymous array

Neither array name nor array size is specified is called as anonymous array.

Syntax:

new <array type>[] {<array initialization block>};

Example:

new int[] {1,2,3,4};

Usage: Anonymous arrays are most offenly used in method calling.

Example:

add(new int[]{1,2,3}, new int[]{4,5,6}); **//method calling**

length field vs length() method

The 'length' field is used to get array size.
The 'length()' method is used to get string size.

Example:

```
int arr[] = {1,2,3,4};
SOP(arr.length); // 4

String str = new String("ASPIRE");
SOP(str.length()); // 6
```

Disadvantages with arrays:

1) The size of an array is fixed i.e., size of an array is neigher increased nor decreased.
2) The elements in an array are homogeneous.
3) There is no pre-defined data structure existing to manage array data.

Conclusion: In order to address above three array limitations, Collection API was introduced.

Main method

```
public static void main(String[] args){}
```

Accessibility modifier	Used to invokes method before object created.	Return type	Method name	Parameter type

Command Line Arguments

The arguments which are passing from the command prompt to the main(String[] args) method is called as command line arguments.

Example:

```
java Addition  1   2

                args[0]        args[1]
```

Example:

//Addition.java

```
class Addition{
public static void main(String[] args){
        int a = Integer.parseInt(args[0]); //"10" → 10
        int b = Integer.parseInt(args[1]); //"20" → 20
        int sum = a + b;
        System.out.println("Total="+sum);
}
}
```

Java Addition 10 20

O/P:

Total=30

OPERATORS and CONTROL FLOW

- ***Arithmetic Operators***
- ***Increment or Decrement Operator***
- ***Relational Operators***
- ***Equality Operators***
- ***Logical boolean operators***
- ***Short-circuit(or conditional) operators***
- ***Ternary operator***
- ***Assignment Operators***
- ***Type Casting***
- ***Selection Statements***
- ***Iteration Statements***
- ***Transfer Statements***

Arithmetic Operators

The arithmetic operators are +, -, *, /, % (Remainder).

Numeric Promotion in Arithmetic Expressions

Floating point arithmetic is performed if any of the operand type is floating-point type, otherwise, integer arithmetic is performed.

Operand1	Operand2	Promoted type
Byte	Byte	Int
Short	Short	Int
Char	Char	Int
Int	Long	Long
Int	Float	Float
Long	Float	Float
Double	Any type	Double

//Integer Arithmetic

//Floating point Arithmetic

In arithmetic expression, if the operand type is byte, short, or char, then they are automatically promoted toint.

Increment (++) or Decrement(--) operator

These are unary operators. Depending on the operator used, the value of the variable is either incremented or decrement by 1.

Pre increment

++i adds 1 to the value in 'i', and stores the new values in i. It returns the new value of the expression. It is equivalent to the following statement.

```
i=i+1
int result = i;
return result;
```

Post incremet

j++ adds 1 to the value in j, and stores the new values in j. It returns the old value of the expression. It is equivalent to the following statement.

```
result=j;
j=j+1
```

return result;

Pre decrement

--i subtracts 1 to the value in i, and stores the new values in i. It returns the new value of the expression. It is equivalent to the following statement.

```
i=i-1;
result = i;
return result;
```

Post decrement

j-- subtracts 1 to the value in j, and stores the new values in j. It returns the old value of the expression. It is equivalent to the following statement.

```
result=j;
j=j-1
return result;
```

int a = ?;
int b = ?

Expression	Initial value of a	Final value of a	Final value of b
b = ++a;	6	7	7
b = a++;	6	7	6
b = --a;	6	5	5
b = a--;	6	5	6
b = ++a + --a;	6	6	13
b = a++ + a--;	6	6	13
b = a++ + --a	6	6	12
b = a + ++a;	4.5	5.5	10.0
Char a = 'y'; Char b = a++;	'y'	'z'	'y'
b= ++(++a)	6	Compilation Error	Compilation Error

Relational operators

The relational operators are **<, <=, >, >=**. All relational operators are binary operators. The evaluation results in a **boolean value**.

Example:

```
System.out.println(1< 2);  //true
```

Equality Operators (==, !=)

Equality operators are evaluated to **boolean value.**

Example:

```
System.out.println( 1 == 1); //true
System.out.println(true == true); //true
System.out.println((flase == false)); //true
System.out.println((true == false)); //false
System.out.println((true != false)); //true
System.out.println((null == null));  //true
```

Boolean logical Operators (&, |, ^, !)

The boolean logical operators are applicable only for **boolean operands**, and returning a boolean value.

& → Logical AND
| → Logical OR
^ → Logical XOR
! → Logical NOT

! → Logical Complement

Truth table for boolean logical operators.

boolean X = true;
boolean Y = true;

X	Y	!X	X&Y	X|Y	**X^Y**
True	**True**	False	True	True	False
True	**False**	False	False	True	**True**
False	**True**	True	False	True	**True**
False	**False**	True	False	False	False

For boolean logical AND, OR, and XOR operators, **both operands are always executed**.

Example:

```
Int i = Integer.parseInt(args[0]);//"10" → 10
if( i > 10 & i++ < 20){}          // 'i' is incremented irrespective of first condition.
```

Short-curcuit (or conditional) operators (&&, ||)

&& → Conditional AND

|| → Conditional OR

Truth table for conditional operators:

X	Y	X && Y	X || Y
True	True	True	**True**
True	False	False	**True**
False	True	**False**	True
False	False	**False**	False

For Conditional AND (&&) operator, the second operand never evaluated if the first operand is false.

For Conditional OR (||) operator, the second operand is never evaluated if the first operand is true.

Logical Operators (&, |)	**Conditional Operators (&&, ||)**
The second operand is always evaluated. I.e., both operanads are always executed.	The second operand is optionally executed i.e., if the first operanad determines the result, then second operand never executed.

Example:

```
Int a = Integer.parseInt(args[0]);
Int b = Integer.parseInt(args[1]);
if((b!=0) && (a%b==0)){ //Never throws ArithmeticException
        System.out.println("No exception");
}
```

Ternary Operator (?:)

It is used to decide between two actions.

syntax:

<data type> <variable name> = <condition> **?** <expression1> **:** <expression2>;

If the <condition> is true then <expression1> is evaluated; otherwise, <expression2> is evaluated. The

<expression1> and <expression2> must be evaluate to values of compatible types with left hand side declaration.

Example: Find out max number

```
int a = Integer.parseInt(args[0]); //"10" → 10
int b = Integer.parseInt(args[1]);//"20"→ 20
int max = a>b?a:b;
```

The ternary operator is equivalent to if-else but additionally returns value.

Assignment Operators

There are three types of assignment operators:

- Simple assignment operator
  Exmaple:
  ```
  int a = 10;
  ```

- Chained assignment operator
  Example:
  ```
  int a, b,c;
  a = b = c =10;
  ```

- Compound assignment operator
  The following are the possible compound assignment operators: +=,-=,*=,/=,%=,&=,|=,^=
  Example:
  ```
  sum  +=  x; //which is equivalent to sum = sum + x;
  ```

Type casting

Primitive or Reference type widening does not require explicit cast.
But, either Primitive or Reference type narrowing requires explicit cast.
Example:

```
float f = 10.10F;
//int i = f; //Compilation Error saying “Incompatible types”
Int i = (int)f; // To avoid compilation error, use type casting
```

CONTROL FLOW

Control flow statements govern the flow (order) of control (execution) in a program during execution. There are three main categories of control flow statements:

- ✓ **Selection statements:**
  - if
  - if-else
  - else if ladder
  - switch
- ✓ **Iteration statements:**
  - while loop
  - do-while loop
  - Basic for loop
  - Enhanced for loops (JDK1.5)
- ✓ **Transfer statements:**
  - break
  - continue
  - return
  - try-catch-finally
  - throw
  - assert

Selection statements

The selective statements are used to execute group of statements (block of statements) at most once i.e., zero or once.

I. if() statement

It is used to decide whether block of statements are executed or not based on the <condition>.

Syntax:

```
if(<condition>){
        <statement(s)>
}
```

The <condition> must be **boolean value** but not numeric number. The if block will be executed only if the <condition > is true.

Example:

```
Int age = Integer.parseInt(args[0]);
If(age < 18){
        System.out.println("Your age does not permit to login to our website.");
}
```

II. if-else

The if-else statement is used to decide between two actions.

Syntax:

```
If(<condition>){
        <statement(s)>
}else{
        <statement(s)>
}
```

If the <condition > is evaluated to true then if block is executed, otherwise else block is executed.

Example:

```
int age = Integer.parseInt(args[0]);
If(age < 18){
```

```
        System.out.println("Your age does not permit to login to our website.");
}else{
        SOP("Logged into Inbox");
}
```

III. else if ladder

It is used to select one among many alternatives.

Syntax:

```
If(<condition1>){
        <statement(s)>
}else if(<conditon2>){
        <statement(s)>
}
…
else{
        <statement(s)>
}
```

The final else block is executed only if none of the above <condition>s are true.

Example:

```
if(false){

}else if(false){

}else if(false){

}else{                          //Else block is executed since all above if() blocks are evaluated to false.

}
```

IV. switch

Switch statement is used to choose one among many alternative actions.

Syntax:

```
switch(<switch expression>){
        case label1: <statement(s)>
        case label2: <statement(s)>
        …
        case labeln: <statement(s)>
        default:<statement(s)>
}
```

The valid <switch expression> types are:

- ✓ byte, short, int, char, but not long
- ✓ Byte, Short, Integer, Character, but not Long

The valid case label types are:

- ✓ byte, short, int, char values but not long value, but not any wrapper classes.

Program: Display Day name

```
class SwitchDemo{
  public static void main(String[] args){
        int day = Integer.parseInt(args[0]);
        switch(day){
```

```
                case 1: System.out.println("Monday"); break;
                case 2: System.out.println("Tuesday"); break;
                case 3: System.out.println("Wednesday"); break;
                case 4: System.out.println("Thursday"); break;
                case 5: System.out.println("Friday");  break;
                case 6: System.out.println("Saturday");break;
                case 7: System.out.println("Sunday"); break;
                default: System.out.println("Valid options are: (1-7)");
        }
  }
}
```

Java SwitchDemo 3

o/p:

Wednesday

Iteration Statements

The iterate statements are used to execute block of statements repeatedly until the boolean <condition> becomes false.

I. While loop

In case of while loop, the <condition> will be evaluated before executing body.

Syntax:

```
while (<condition>){
        < body>
}
```

The <condition> is always boolean value.

Program: Write a program to print numbers from 1 to 10.

```
Int I =1;
While(I <= 10){
        System.out.print(I+"\t");
        I++;
}
```

II. do-while loop

In case of do-while, the body is executed before <condition> is evaluated i.e., the body will be executed atleast once.

Syntax:

```
do {
        <body>
} while (<condition>);          //Watch: Semicolon at the end.
```

Program: Write a program to add numbers from 1 to 10;

```
Int I = 1;
Int sum = 0;
Do{
        Sum  += I;
        I++;
}while(i<=10);
```

Difference between while and do-while

While	Do-while

The <condition> is evaluated before executing the body. Hence, while loop executes zero or more times.	The body will be executed before the <condition> is evaluated. Hence, the body would be executed one or more times **i.e., body will be executed at least once**.

III. Basic for loop (Simple or General for loop)

This loop is best suitable if we know the number of iterations in advance.
For example, reading array elements or reading collection elements.

Syntax:

```
for (<initialization>; <condition>; <increment or decrement>){
        <body>
}
```

Program: Write a program to add array elements using for loop.

```
Int[] arr = {10,20,30,40};
Int sum = 0;
for(int i=0; I < arr.length; i++){
        Sum += arr[i];
}
Sop("The total="+sum);
```

IV. Enhanced for (or for-each) loop

This is first time introduced in **jdk1.5**. **This is the most convenient loop for retrieving elements from array or collection.**
This loop cannot be used for general purpose.
This loop iterate elements always in forward direction but not in reverse direction.

Syntax:

```
for(<element declaration> : <array or collection name>){
        <body>
}
```

Where type of <element declaration> is same as array type or collection type.

Example:

Int[] arr = {10,20,30};

Basic for loop	Enhanced for loop

for(int I = 0; I < **arr.length;** I++){
SOP(arr[i]);
}
The basic for loop uses array **index** to read array data.
	for(**int** e : arr){
	SOP(e);
	}
	The Enhanced for loop directly uses **element** but not index. Hence, simple to use.

Note:

1. Enhanced for loop is used only with arrays and collections but not for general purpose.
2. Enhanced for loop cannot be used to retrieve elements in reverse order.

Transfer Statements

I. break

The '**break'** is used to come out of the loops.
We can use break statement in one of the following cases:

- Inside loops
- Inside switch

Example:

```
For(int I=0 ; I<10;  i++){
        For(int j=0; j<10; j++){
                If(i==j)   break;  //the nearest for loop will be terminated.
        }
}
```

II. Continue

The continue statement is used to transfer the execution back to the start of the loop i.e., the remaining statements after continue will be skipped.

Program: Write a program to add even numbers but print odd numbers.

```
Int sum = 0;
for(int i = 0; i<=10; i++ ){
        if(i%2 == 0){  //even number
                sum+= I;          //Adding even numbers
                continue;
        }
        System.out.print (i+"\t");          //Printing odd numbers
}
Sop("The sum of even numbers is:"+sum);
```

III. return

The return statement is always used from method block. The return statement is used to stop execution of a method and transfer control back to the method calling.

There are two forms of the return statement:

return; //Optionally used with methods whose return type is void **or** from constructor.

return <expression>; //Must be used with methods whose return type is other than void.

A void method need not have a return statement – in which case the control normally returns to the caller after the last statement in the method's body has been executed. However, a void method can optionally specify the first form of the return statement – in which case the control immediately transferred to the method caller.

Program: Write a program to return maximum number

```
int max(int x, int y){
        If(x > y)
                Return x;
        else
                Return y;
}
```

(or)

```
int max(int x, int y){
        return x>y?x:y;  //ternary operator
}
```

ACCESS CONTROL

- ***Package declaration***
- ***Modifiers***
- ***Accessibility modifiers***
- ***import***
- ***Java Progamming Structure***
- ***Generate Java API Documentation***

Packages

In java, package is a **directory structure** which contains group of related classes, interfaces, enums, exceptions, errors, annotations, or subpackages.

Example:

Package	Directory	Types
java.lang	java\lang	String, System, Object, Integer, etc
java.io	java\io	InputStream, OutputStream, Reader, Writer, etc
java.util	Java\util	List, Set, Map, Vector, Hashtable, etc

Packages are declared using **package** keyword:

Format:

package <domain name> <.company name><.project name><.module name>;

Example:

```
package edu.aspire.accounting.billing; //multi level package
package edu.aspire.accounting.core; //multi level package
```

At most one package declaration can appear in a source file, and it must be the first statement in the source file.

Example:

```
package aspire;
import java.lang.String;        //Correct
```

Use **'–d'** option to compile classes with package declaration.

Syntax:

Javac **-d <directory>** <java source file>

Example:

Javac **–d .** *.java

where '.' Specifies current directory, where the generated class files will be placed.

Example:

```
package aspire;
class Test{
        public static void main(String[] args){
                System.out.println("Package concepts");
        }
}
```

Compilation:

Javac -d . Test.java

Also, the **fully qualified type** name must be specified in the java command.

Example:

Java aspire.Test

To access package types from other packages, use either fully qualified type (<package_name>.<type>) or import types using 'import' keyword.

Modifiers in java

There are 13 modifiers in java:

Private
default
Protected
Public
Final
Static
Abstract
Interface
Synchronized
Transient
Native
Strictfp
volatile

Accessibility Modifiers

In java, there are 4 accessibility modifiers: private, no (or default or package) modifier, protected, and public.

The below table summarizes the **scope** of accessibility modifiers:

Scenario	Private	default	Protected	Public
Within the same class	Yes	Yes	Yes	Yes
Subclass of the same package	No	Yes	Yes	Yes
Non subclass of the same package	No	Yes	Yes	Yes
Subclass from different package	No	No	Yes	Yes
Non subclass from different	No	No	No	Yes

Note:

1) The scope of the private members is anywhere within the class.
2) The scope of the default members is anywhere within the same package.
3) The scope of the protected members is anywhere with in the same package as well as only subclass of the different package.
4) The scope of the public members is anywhere**!**
5) The default modifier is also know as **package** modifier.

Imports

The given class can be accessible from outside the package using fully qualified type name, however, writing long names can become tedious.

Example:

```
edu.aspire.One obj = new edu.aspire.One();
```

The import facility in java makes it easier to access classes from different packages, which will be declared once using import statement and used it for any number of times just with simple type name.

Example:

```
import edu.aspire.One;
One obj = new One();
```

Java Programming Structure

The java program is basically divided into following sections:

[Comments Section]

[Package Declaration]

[Import Statement(s)]

<class modifiers> class <class name> [extends clause] [implements clause]{

[fields]

[constructors]

[methods]

}

Comments

Java supports 3 types of comments:

1) Multiline comments (From **C** language)

   syntax:

   ```
   /*
      ….
      ….
   */
   ```

2) Single Line comment (From **C++** language)

   Example:

   //Tomorrow is a holiday

3) **Java doc comments**

   This is newly added in JAVA. These comments are used to add description about class as well as methods.

   Syntax:

   ```
   /**
   *
   */
   ```