Java means DURGA SOFT..

CORE JAVA Material

India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

Language Fundamentals:

To design Java applications, Java has provided some building blocks in the form of the following Language Fundamentals.

1.Tokens

2.Data Types

3.Type Casting

4.Java Statements

5.Arrays

1.Tokens

- The smallest logical unit in Java programming is called as "Lexeme".
- The collection of "Lexemes" come under a particular group called as "Token".

To design Java Applications,Java has provided the following tokens:

1.Identifiers

2.Literals

3.Keywords/Reserved words

4.Operators

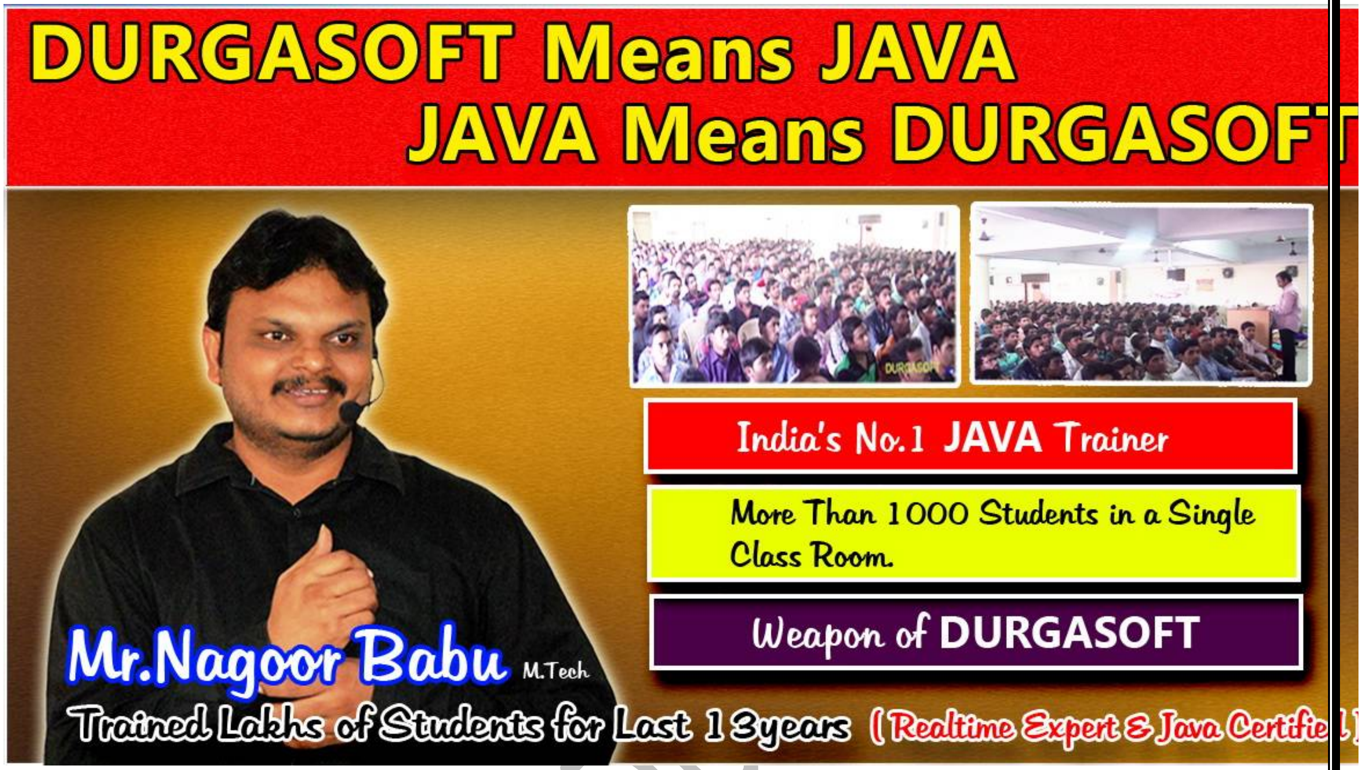
1.Identifiers:

- Identifier is a name assigned to the Java programming constructs like variables,methods,classes,interfaces......
- To provide identifiers in Java programs we have to use the following rules and regulations.

1.Identifiers should not start with a number,identifiers may start with an alphabet,_symbols,$ symbols but the subsequent symbols may be a number,_symbols,$ symbols,an alphabet.

Ex:

```
int eno=111; ---> valid

String _empAddr="Hyd"; ---> Valid

float $empSal=50000.of; --->valid

String 9eid="999"; ---->InValid

int emp9No=999; ---->Valid

String emp_Addr="Hyd"; -->Valid
```


DURGASOFT Means JAVA
JAVA Means DURGASOFT
India's No.1 JAVA Trainer
More Than 1000 Students in a Single Class Room.
Weapon of DURGASOFT
Mr.Nagoor Babu M.Tech
Trained Lakhs of Students for Last 13years (Realtime Expert & Java Certified)

2.Identifiers will not allow all the operators,the special symbols like #,@,.,:,.......except $ and_ symbols

Ex:

```
int emp+No=111; --->Invalid

String #empAddr="Hyd"; --->Invalid

String emp@Hyd="Durga"; --->Invalid

String emp-Name="Durga" --->Invalid

String emp_Name="Durga"; --->valid
```

3.Identifiers will not allow spaces in the middle.

Ex:

```
forName(); -->Valid

for Name(); -->InValid

get Input Stream(); -->InValid

getInputStream(); -->Valid
```

4.Identifiers should not be duplicated with in the same scope but identifiers may be duplicated in two different scopes.

Ex:

```
class A{

int i=10;

float i=22.22f;--->Error

double f=23.32;

void m1(){

long i=20;

float f=10.0f;

byte f=23; ---->Error

}}
```

NOTE:In java,all the instance varaibles will be stored in "Heap Memory",all the local varaibles will be stored in "Stack Memory" and all the static variables will be stored in "Method Area".

5.In java applications,all the predefined class names and interface names are able to use as identifiers.

Ex:

```
class Test{

public static void main(String args[]){

int Exception=10;

System.out.println(Exception);

}}

class Test{

public static void main(String args[]){

float Thread=22.22f;

System.out.println(Thread);

}}

class Test{

public static void main(String args[]){

String String="String";

System.out.println(String);

}}

class Test{

public static void main(String args[]){

int System=10;

System.out.println(System);

}}
```

Status:Compilation Error,int cannot be dereferenced

Reason:In java program,once if we declare any predefined class name as an integer variable then compiler and JVM will treat that predefined class name as an integer only in the remaining program,it can not be used as class.

In the above context,if we want to use System as class then we have to provide that predeifined class [System] as fully qualified name.

NOTE:Specifying class names along with their package names is called as Fully Qualified Name.

Ex:

java.lang.System

java.util.ArrayList

java.io.BufferedReader

```
class Test{

public static void main(String args[]){

int System=10;

java.lang.System.out.println(System);

System=System+10;

java.lang.System.out.println(System);

}}
```

Along with the above rules and regulations,JAVA has given the following suggestions to declare and use identifiers.

1.Identifiers should be meaning ful,they should reflect particular meaning.

Ex:

String x="abc123";---->Not Suggestible

String accNo="abc123" --->Suggestible

2.In java applications,there is no length restriction for the identifiers but is is suggestible to manage the length of the Identifiers around 10 Symbols.

Ex:

String temporaryemployeeaddress="Hyd";---->Not Suggestible

String tempEmpAddr="Hyd";----> Suggestible

3.If we have multiple words with in a single identifier then it is suggestible to separate multiple words with special notations like '_' symbol.

Ex:

String tempEmpAddr="Hyd";---->Not Suggestible

String temp_Emp_Addr="Hyd";----> Suggestible

2.Literals:

- Literal is a constant assigned to the variables.

Ex:

int a=10;

int--->data type

a----->variable

= ---->operatot

10 --->constant[literal]

; --->special symbol/terminator

To design Java applications,Java has provided the following literals.

1.Integer Literals/Integer Literals:

byte,short,int,long ---->10,20......

char ---->'a','b'.....

2.Floating point Literals:

float ---> 11.11f,12.234f,...............

double --->12.23,23.345..................

3.Boolean literals:

boolean --->true,false

4.String Literals:

String --->"abc","xyz"...

Number systems in Java:

To represent number in Java,we are able to use the following number system.

1.Binary Number System[Base-2]

2.Decimal Number System[Base-10]

3.Octal Number System[Base-8]

4.HexaDecimal Number System[Base-16]

- In Java,the default number system will be Decimal number system.
- If we want to represent a number in binary number system then we have to prepare the number by using the Symbols like '0's and '1's and the number must be prefixed with either "0b or 0B"

Ex:

int a=10;--->Invalid,not a binary number

int b=0b10;-->Valid

int c=0B10;-->Valid

int d=0b102;-->Invalid

NOTE:Binary number system is not possible upto JAVA6 versions it is possible in JAVA7 version and above,because,Binary number System is a new features in JAVA7 version.

- In Java applications,if we want to represent any number in octal number system then we have to prepare the number by using the symbols like 0,1,2,3,4,5,6 and 7 but the number must be prefixed with '0'[zero]

Ex:

int a=10;--->Invalid,it is not octal number

int b=010;--->Valid

int c=0345;-->Valid

int d=0678;-->Invalid

NOTE:Octal number System is supported by both JAVA6 and JAVA7 versions.If we want to represent any number in Hexadecimal number system then we have to

prepare the number by using the symbols like 0,1,2,3,4,5,6,7,8,9,c,d,e and f but the number must be prefixed with either '0x' or '0X'.

Ex:

int a=10; --->Invalid,it is not a hexadecimal number.

int b=0x56 -->Valid

int c=0x789 -->Valid

int d=0xbcde; -->Valid

int e=0xdefg; -->Invalid.

NOTE:Hexadecimal number system is supported by both JAVA6 and JAVA7 versions.

- To improve readability of literals,JAVA7 version is allowing '_' symbols in the middle of the literals.

Ex:

```
long l=1_23_45_677;

System.out.println(l);

O/P:12345677
```

If we compile the above code then compiler will remove all '_' symbols from the number and it will process that number as original number only.

3.Keywords/ReserveWords:

- Keyword is a predefined,which must have both word recognization and basic functionality.
- Reserved word is a predefined word,which must have only word recognization without basic functionality.
- To design Java applications,Java has given the following list of keywords and reserved words.

CORE JAVA with
OCJP/SCJP
JAVA CERTIFICATION
SATISFACTION 100% GUARANTEED
Mr. DURGA M.Tech
JAVA EXPERT
Trained Thousands of Students
One to One
VIDEO
CLASSES
EVERYTHING AT YOUR CONVENIENC
At your convenient Time
With in your convenient duration
AN ISO 9001:2008 CERTIFIED
DURGA
Software Solutions®
www.durgasoft.com
# 202, 2nd Floor, HUDA Maitrivanam,
Ameerpet, Hyd. Ph: 040-64512786,
9246212143, 8096969696

Keywords:

1.Datatypes and return Types:

Byte,short,int,long,float,Double,boolean,char,void.

2.Access Modifier:

public,protected,private,static,final,abstract,native,volatile,

strictfp,transient,synchronized..

NOTE:The keywords public,private,protected are not access specifiers,there are come under access modifiers only,because,in Java programming languages no features like access specifiers.

3.Flow controllers:

if,else,switch,case,default,break,continue,return,for,while,do..

4.class/object related:

class,extends,interface,implements,enum,this,super,new,package,import......

DURGASOFT Means JAVA
JAVA Means DURGASOFT
India's No.1 JAVA Trainer
More Than 1000 Students in a Single Class Room.
Weapon of DURGASOFT
Mr.Nagoor Babu M.Tech
Trained Lakhs of Students for Last 13years (Realtime Expert & Java Certified)

5.Exception Handling:

throw,throws,try,catch,finally

Reserved Words:

goto,const.

Operators:

- Operator is a symbol it able to perform a particular operation over the provided operands.

To design Java applications java has provided the following list of operators.

1.Arithmetic Operators:

+,-,*,/,%,++,--

2.Assignment Operator:

=,+=,-+,/+,%=

3.Comparsion Operators:

==,!=,<,><=,>=

4.logical Boolean Operators:

&,|,^,............

5.logical Bitwise Operators:

&,|,^,<<,>>,.............

6.Ternary Operator:

exp1?exp2:exp3;

7.Shortcircuit Operator:

&&,||

Ex:1

```
class Test{
public static void main(String args[]){
int a=10;                                    O/P:
System.out.println(a);                        10
System.out.println(a++);                      10
System.out.println(++a);                      12
System.out.println(a--);                      11
System.out.println(--a);                      11
System.out.println(a);                        10
}}
```

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

```
int a=5;

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

}}
```

```
class Test{

public static void main(String args[]){

int a=7;

System.out.println(++a+--a*++a);

}}
```

```
class Test{

public static void main(String args[]){

int a=4;

System.out.println(a++-++a*++a+--a);

}}
```

```
class Test{

public static void main(String args[]){

int a=6;
System.out.println((++a+--a)*(--a+--a)+(++a-++a)*(--a-a--));

}}
```

Boolean Operator:

```
            boolean b1=true;

            boolean b2=false;

        System.out.println(b1&b1);  //true

        System.out.println(b1&b2);  //false

        System.out.println(b2&b1);  //false

        System.out.println(b2&b2);  //false

        System.out.println(b1|b1);  //true

        System.out.println(b1|b2);  //true

        System.out.println(b2|b1);  //true

        System.out.println(b2|b2);  //false

        System.out.println(b1^b1); //false

        System.out.println(b1^b2);  //true

        System.out.println(b2^b1); //true

        System.out.println(b2^b2);  //false
```

A	B	A&B	A|B	A^B
T	T	T	T	F

T	F	F	T	T
F	T	F	T	T
F	F	F	F	F

A	B	A&B	A|B	A^B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Bitwise Operator:

```
int a=10;

int b=2;

System.out.println(a&b);  --->2

System.out.println(a|b);  --->10

System.out.println(a^b);  --->8

System.out.println(a<<b); --->40

System.out.println(a>>b); --->2
```

Short Circuit Operators:

- In the case of boolean "or" operator,if the first operand is true then it is possible to predict the over all expression result is "true" with out checking second operand.

- In java applications,in the case of "|" and "&" boolean operators,even though the first operand value is sufficient to predict the overall expression result still JVM will execute second operand,here,it is unnecessary to execute second operand value,it will reduce the performance of the application.
- In the above context,to improve the performance of the applications, we have to use ShortCircuit operators like "||" and "&&".
- In the case of short circuit operators,if the first operand value is sufficient to predict overall expression result then JVM will not execute second operand value,this approach will increase the application performance.
- In the case of "||",if the first operand value is true then JVM will not check second operand value and it will predict overall expression result is true.
- In the case of "&&",if the first operand value is false then JVM will not check second operand value,JVM will predict the overall expression result is false.

Ex:

```
 class Test{
public static void main(String args[]){
int a=10;
int b=10;
if((a++==10) | (b++==10)){
System.out.println(a);
System.out.println(b);
}
int c=10;
int d=10;
if((c++==10) || (d++==10)){
```

```
System.out.println(c);
System.out.println(d);
}}}
Ex:
class Test{
public static void main(String args[]){
int a=10;
int b=10;
if((a++!=10) | (b++!=10)){
System.out.println(a);
System.out.println(b);
}
int c=10;
int d=10;
if((c++!=10) || (d++!=10)){
System.out.println(c);
System.out.println(d);
}}}}
```

DataTypes:

- Java is Strictly a typed programming language,where in java applications,before going to represent the data first we have to decide which type of data we are representing.To decide the type of data which we are representing we have to use "data types".

In java applications,data types are able to provide the following advantages:

1.We can identify memory sizes for the data to store.

2.We can identify the range of values which we want to assign to the varaibles.

Note:Memory sizes for the primitive data types is fixed in Java irrespective of the operating system which we used,this nature of the data types will make java as a platform independent programming language.

```
int i=10;---->valid
    a=20---> invalid
int i;
    i=10;---->Valid
```

To desgin java applications,java has provided the following list of datatypes:

1)Primitive Data Tyeps/Primary data types

1)Numeric Data Types:

1)Integral data Types/Integer data Types

Size	Default	value
byte	1 byte	0
short	2 bytes	0
int	4 bytes	0
long	8 bytes	0

2)Non-Integral Datatype/Floating point Datatypes

Size	Default	value
float	4 bytes	0.0f
double	8 bytes	0.0

2.Non-Numeric Data Types:

Size	Default	value
Char	2 bytes	“[singlespace]
boolean	1 bit	false

2.User defined Datatypes/Secondary Datatypes

- All classes,All Interfaces,All Arrays......If we want to identify the range values of each and every datatypes then we have to use the following formula

-2^{n-1} to $2^{n-1} - 1$

where n is no.of bits

Ex:

byte data type ---> 1 byte --> 8 bits

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

7 7

-2 to 2 -1

-128 to 128 -1

-128 to 127

Calculate min&max long data type exact value

long data type --> 8 bytes -->64 bits

64-1 64-1

-2 to +2 - 1

- To get the minimum value and maximum value of each and every primitive data type,we have to use "MIN_VALUE" and "MAX_VALUE" constants from each and every Wrapper class.

Note:Wrapper classes are the classes representation of primitive datatypes.

Primitives	Wrapper classes
Byte	Byte
Short	Short
Int	Integer
long	Long
Float	Float
Double	Double

char	Character
Boolean	Boolean

```
class Test{

public static void main(String args[]){

System.out.println(Byte.MIN_VALUE+" -->"+Byte.MAX_VALUE);

System.out.println(Short.MIN_VALUE+"-->"+Short.MAX_VALUE);

System.out.println(Integer.MIN_VALUE+"--->"+Integer.MAX_VALUE);

System.out.println(Long.MIN_VALUE+"--->"+Long.MAX_VALUE);

System.out.println(Float.MIN_VALUE+"--->"+Float.MAX_VALUE);

System.out.println(Double.MIN_VALUE+"--->"+Double.MAX_VALUE);

}}
```

TypeCasting:

- The process of converting the data from one data type to another data type is called as TypeCasting.

  There are two types of Type Castings:

  1)primitive datatypes casting.

  2)userdefined datatypes casting

- The process of converting the data from one user defined data Type to another user defined data Type is called as User defined data types Type Casting.
- To perform User Defined data types type casting,we must provide either "extends" relation or "implements" between two user-defined data types.

Primitive data types type Casting:

- The process of converting the data from one primitive data type to another primitive data type is called as Primitive data types type casting.

  There are two types of primitive data types type casting.

  1.Implicit type casting.

  2.Explicit type casting.

1.Implicit Type Casting:

- The process of converting the data from lower datatype to higher datatype is called as Implicit Type Casting.
- To cover all the possibilities of implicit type casting,Java has provided a predefined chart.
- To perform implicit type casting in Java,simply we have to assign lower data type variable to higher data type variable.

```
byte b=10;

int i=b;
```

- when we compile the above code,compiler will check right side variable[b] datatype is compatible with left side variable data type for = ,if right side variable data type is not compatible with left side variable data type then compiler will rise an error like "Possible loss of precision".If right with left side variable datatype then compiler will not rise any error.

Note:In Java,all lower datatypes are compatible with higher datatypes but higher datatypes are not compatible with lower data types.Due to this reason,we can assign lower dataTypes to higher data types directly.

When we execute the above code then JVM will perform the following two actions.

1.JVM will convert the right side data type to left side data type implicitly and automatically.

2.JVM will copy the value from right side variable to left side variable.

```
byte b=10;

int i=b;
```

status:No Compilation error

o/p:10 10

```
byte b=65;

char c= b;
```

status:Compilation Error,possible loss of Precision

Reason:No conversions are existed between byte and char,short and char as per implicit type casting chart.

```
float f=22.22f;

long l=f;
```

Status:Compilation Error,possible loss of Precision

```
long l=10;

float f=l;
```

Status:No Compilation error

o/p 10   10.0

```
float f=22.2f;
```

long l=f;

Status:Compilation Error,possible loss of Precision

Reason:Float datatype is taking 4 bytes of memory and long datatype is taking 8 bytes of memory.

- Float datatype is able to store more data when comapred with long datatype as per their internal data arrangement.Therefore,float is higher data types when compared with long data types.

byte b=128;

Status:Compilation Error,possible loss of Precision

Reason:If assign a value to a particular datatype variable which is more than the maximum limit of the respective data type then compiler will treate that value is of the next higher data type so compiler will rise an error.

Note:For byte and short the next higher data type is "int".

byte b1=10;

byte b2=20;

byte b=b1+b2;

Status:Compilation Error,possible loss of Precision

Reason:X,Y and Z are primitive data types

X+Y=Z

Where if X and Y belongs to {byte,short,int} then Z should be "int".

Whether if either X or Y both belongs to {long,float,double} then Z should be max(X,Y) as per implict type casting chart.

Ex:

byte+byte=int

byte+short=int

short+int=int

int+long=long

long+float=float

float+double=double

```
short s=10;

byte b=20;

int i=s+b;

System.out.println(i);
```

Explicit Type Casting:

- The process of converting the data from higher data type to lower data type is called as Explicit type casting.

To perform explicit type casting,we have to use the following pattern.

```
P a=(Q) b;
```

where P and Q are primitive data types and Q must be either same as P or lower than P as per implicit type casting chart.

Ex:

```
  int i=10;

  byte b=(byte)i;

  System.out.println(i+"     "+b);
```

Status:No compilation Error

o/p:10 10

- When we compile the above code then compiler will check whether right side data type and left side datatype are compatible or not,if not compiler will rise an error like "Possible loss of Precision".
- If both the datatypes are compatible then compiler will not rise any error.
- In the above example,compiler will not rise any error because,both left side data type and right side data type are "byte" data types.
- When we execute the above code,JVM will perform the following actions.

1.JVM will convert the right side variable data type to the datatype which we specified inside braces[()] i.e cast operator.

2.JVM will copy the value from right side variable to left side variable.

Ex:

```
byte b=65;

char ch=(char)b;

System.out.println(b+"     "+c);
```

Status:No Compilation Error.

O/P:65 A

```
char c='B';
```

```
short s=(Short)c;  or short s =(byte)c;

System.out.println(c+"  "+s);
```

Status:No Compilation Error

O/P:B   66

Note:In the case of implicit type casting,conversions are not existed between char and byte,char and short but in the case of explicit type casting conversions are possible between char and byte,char and short.

Ex:

```
float f=22.22f;

long l=(long)f; or long l=(int)f;

System.out.println(f+"    "+l);
```

Status:No Compilation Error

O/P:22.22 22

Ex:

```
byte b1=10;

byte b2=10;

byte b=(byte)(b1+b2); or byte b=(byte)b1+b2;
```

[Status:Compilation Error,possible loss of Precision]

```
System.out.println(b);
```

Status:No Compilation Error

O/P:20

Ex:

```
float f1=10.0f;

long L1=10;

long L=f1+L1;  or long L=(int)f1+L1;

System.out.println(L);
```

Status:No Compilation Error

O/P:20

Ex:

```
double d=22.22;
byte b=(byte)(short)(int)(long)(float)d;
System.out.println(b);
```

Status:No Compilation Error

O/P:22

Ex:

```
int i=135;
byte b=(byte)i;
System.out.println(i+"----------------"+b);
```

Status:No Compilation Error

O/P:135 -121

Reason:

```
int i=X;
byte b=(byte)X;
```

If X is 127+n then b should be calculated from the below formula.

```
b=-128+n-1;


Ex:

int i=129;

short s=(byte)i;

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

Status:No compilation Error

O/P:129   -127
```

JAVA Statements:

- Statement is a collection of expressions.

To design java applications,Java has given following list of statements.

1.General Purpose Statements:

Declaring varaibles,methods,classes........

Accessing varaibles,methods...........

Creating objects,packages..............

2.Conditional Statements:

if , switch

3.Iterative Statements:

for,while,do-while

4.Transfer Statements:

break,continue,return

5.Exception Handling Statements:

throw,try-catch-finally

6.Synchronized Statements[Multi threading]

synchronized methods,synchronized blocks

Conditional Statements:

- These are the Java Statements,which will allow to execute a block of Java code under a particular condition.

  There are two types of conditional Statements.

  1.if

  2.switch

1.if:

Syntax-1:

```
if(condition)

{
```

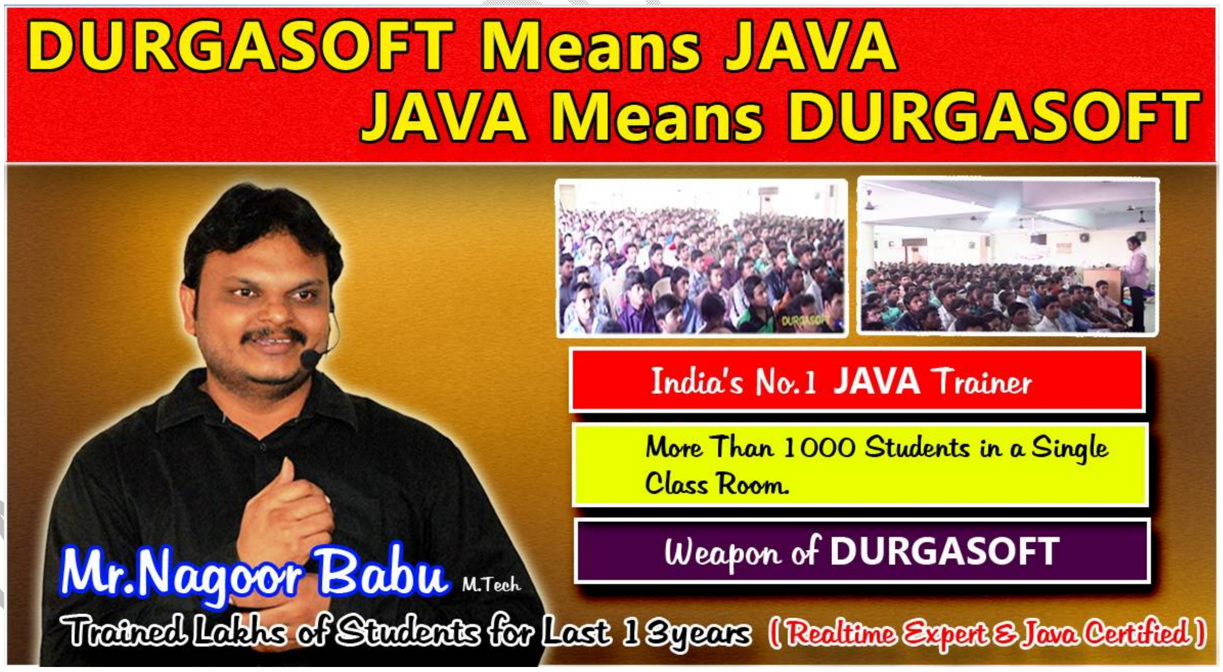
```
----statements-------
}
Syntax-2:
if(condition)
{
----statements----
}
else
{
----statements---
}

Syntax-3:
if(condition)
{
-----statements---
}
else if(condition)
{
----statements---
}
------
------
else{
---statements---
}
```

Ex:

```
int i=10;
int j;
if(i==10){
j=20;
}
System.out.println(j);
```

Status:Compilation Error

```
int i=10;
int j;
if(true){
j=20;
}
```


DURGASOFT Means JAVA
JAVA Means DURGASOFT
India's No.1 JAVA Trainer
More Than 1000 Students in a Single Class Room.
Weapon of DURGASOFT
Mr.Nagoor Babu M.Tech
Trained Lakhs of Students for Last 13years (Realtime Expert & Java Certified)

```
System.out.println(j);
```

Status:No Compilation Error

O/P:20

```
int i=20;
int j;
if(i==10){
j=20;
}
else{
j=30
}
System.out.println(j);
Status:No Compilation Error
O/P:20


final int i=10;
int j;
if(i==10){
j=20;
}
System.out.println(j);
Status:No Compilation Error
O/P:20


int i=10;
int j;
if(i==10){
j=20;
}
else if(i==20){
j=30;
}
System.out.println(j);
```

Status:Compilation Error

1.In Java,only class level varaibles are having default values,local variables are not having default values.

2.If we declare any local variable then we must provide initialization for that local variable either at the same declarative statement or atleast before accessing that local variable.

3.In java applications,complier is able to recoginze and execute constant expressions which are having only constants.

4.In java,compiler is unable to recognize and execute variable expressions which are having atleast one variable.

Switch:

- In Java applications,"if" conditional statement is able to allow single condition checking but switch is able to allow multiple conditions checking.

  In general,Switch will be utilized to prepare menu driven applications.

  Syntax:

```
switch(var){
case 1:
-------statements-----------
break;
------
------
case n:
-------statements--------
break;
default:
---------statements------
break;
}
```

Ex:

```
class Test{
public static void main(String args[]){
int i=10;
switch(i){
case 5:
        System.out.println("FIVE");
        break;
case 10;
        System.out.println("TEN");
        break;
case 15;
```

```
          System.out.println("FIFTEEN");
          break;
     case 20;
     System.out.println("TWENTY");
     break;
     default:
     System.out.println("Default");
     break;
     }}}
```

Rules To write switch:

1.Switch is able to allow the data types like byte,short,int and char as parameter.

Ex:

```
char c='A';
switch(c){
case 'A';
     System.out.println("A");
     break;
case 'B'
     System.out.println("B");
```

```
        break;

        default

        System.out.println("Default");

        break;
```

DURGASOFT Means JAVA
JAVA Means DURGASOFT
India's No.1 JAVA Trainer
More Than 1000 Students in a Single Class Room.
Weapon of DURGASOFT
Mr.Nagoor Babu M.Tech
Trained Lakhs of Students for Last 13years (Realtime Expert & Java Certified)

Ex:

```
long l=10;

switch(l){

---------

}
```

status:Compilation Error

2.JAVA7 is able to allow String data type as parameter,which is not possible upto JAVA6 version.

Ex:

```
String str="BBB";

switch(str){

case "AAA":
```

```
          System.out.println("FIVE");
          break;
case "BBB";
          System.out.println("TEN");
          break;
case "CCC";
          System.out.println("FIFTEEN");
          break;
case "DDD";
          System.out.println("TWENTY");
          break;
          default:
          System.out.println("Default");
          break;
          }}}
```

3.In switch,all the cases are optional,it is possible to write switch with out having cases.

```
          int i=10;
          switch(i){
          default:
          System.out.println("Default");
          break;
          }
          Status:No Compilation Error
          O/P:Default
```

4.In Switch Default is optional,it is possible to write switch even without "default".

Ex:

```
int i=100;

switch(i){

case 50:

          System.out.println("50");

          break

case 60:

          System.out.println("60");

          break;

          }
```

Status:No Compilation Error

O/P:No OutPut

5.In Switch,all the cases and default are optional,it is possible to write switch with out cases and default.

Ex:

```
int i=10;

switch(i){

}
```

6.In switch break statement is optional,if we write switch with out break statement then JVM will execute all the instructions right from matched case untill it encounter either break statement or end of the switch.

Ex:

```
int i=10;

switch(i){

case 5:

          System.out.println("FIVE");

case 10:

          System.out.println("TEN");

case 15:

          System.out.println("FIFTEEN");

case 20:

          System.out.println("TWENTY");

 default:

           System.out.println("Default");


}
```

Status:No Compilation Error

```
O/P:     TEN

          FIFTEEN

          TWENTY

          Default
```

7.In switch,all the case values must be available with in the range of the datatype which we passed as parameter to switch.

```
Ex:

byte b=126;

switch(b)       {

case 125;

         System.out.println("125");

         break;

case 126;

         System.out.println("126");

         break;

case 127;

         System.out.println("127");

         break;

case 128;

         System.out.println("128");

         break;

         default:

         System.out.println("Default");

         break;

}

Status:Compilation Error(Possible loss of precision)
```

8.In Switch,all the case values must be either direct constants or final constants.

```
Ex:

final int i=125,j=126,k=127,l=128;

switch(126){
```

```
            case i:
                    System.out.println("125");
                    break;
            case j:
                    System.out.println("126");
                    break;
            case k:
                    System.out.println("127");
                    break;
            case l:
                    System.out.println("128");
                    break;
                    default:
                    System.out.println("default");
                    break;
    }
```

Iterative Statements:

These are the java statements,which will allow to execute a block of instructions repeatedly under a condition.

There are three types of iterative statements in java.

1.for 2.while 3.do-while

1.for:

Syntax:

```
for(expr1;expr2;expr3){

-----body------------

}
```

Ex:

```
for(int i=0;i<10;i++){

System.out.println(i);

}
```

expr1-----> 1 time

expr2----->11 times

expr3----->10 times

body------> 10 times

Ex:

```
int i=0;

for(;i<10;i++){

System.out.println(i);

}
```

Status:No Compilation Error

O/P:0 to 9

Ex:

```
int i=0;
for(System.out.println("Hello");i<10;i++){
System.out.println(i);
}
```

Status:No Compilation Error

Reason:

In for loop syntax,expr1 is optional,we can write for loop with out expr1,we can write any statement as expr1 including System.out.println(----); method but it is Suggestible to utilize expr1 to declare and initialize loop variables.

Ex:

```
for(int i=0;float f=0;i<10 && f<10.0f;i++,f++){
System.out.println(i+"    "+f);
}
```

Status:Compilation Error

Ex:

```
for(int i=0;int j=0;i<10 && j<10;i++,j++){
System.out.println(i+"    "+j);
}
```

Status:Compilation Error

Ex:

```
for(int i=0;j=0;i<10 && j<10;i++,j++){
System.out.println(i+"    "+j);
}
```

Status:No Compilation Error

Reason:In for loop syntax,expr1 is able to allow atmost one declarative statement,it will not allow more than one declarative statement.

Ex:

```
for(int i=0;;i++){
System.out.println(i);
}
```

Status:No compilation Error

O/P:Infinite loop

Reason:In Java,in for loop,expr2 is optional,if we have not provided expr2 then "for" loop will take true as boolean value in place of expr2 even though the default value for boolean variables is false in Java

Ex:

```
for(int i=0;System.out.println("Hello");i++){
System.out.println(i);
}
```

Reason:In for loop,expr2 is optional,if we want to provide any statement as expr2 then that Statement must be boolean statement,it must generate a boolean value as result.

Ex:

```
System.out.println("Before loop");
for(int i=0;i>=0;i++){
System.out.println("Inside loop");
}
System.out.println("After loop");
```

Status:No Compilation Error

O/P:Before Loop

```
       Inside Loop(Infinite)
```

Ex:

```
System.out.println("Before loop");
for(int i=0;true;i++){
System.out.println("Inside loop");
}
System.out.println("After loop");
```

Status:Compilation Error(unreachable statement)

Ex:

```
System.out.println("Before loop");
for(int i=0;;i++){
```

```
System.out.println("Inside loop");

}

System.out.println("After loop");
```

Status:Compilation Error(unreachable statement)

Reason:In Java,if we provide any statement immediately after infinite loop then that statement is called as "UnReachableStatement".

- In the above context,rising an error or not is completely depending on how much compiler is recognized the provided loop is infinite loop.
- If compiler is recognized the provided loop is infinite loop and if the compiler is identified an instruction immediately after infinite loop then compiler will rise an error like "Unreachable Statement".
- If compiler is not recognized the loop is an infinite loop then there is not chance getting "Unreachable Statement".
- Deciding a loop is infinite loop or not is totally depending on the conditional expression,if the conditional expression is constant expression and it results true then compiler will recognize that loop is an infinite loop.
- If the conditional expression is variables then compiler will not recognize that loop is an infinite loop.

Ex:

```
for(int i=0;i<10;){

System.out.println(i);

i=i+1;

}
```

Status:No Compilation Error

Ex:

```
for(int i=0;i<10;System.out.println("Hello")){

System.out.println(i);

i=i+1;

}
```

Status:No Compilation Error

Reason:In for loop,expr3 is optional,we can write for loop with out expr3,in place of expr3 we can write any statement but we will utilize expr3 to perform loop variables incrementation or decrementation operations.

Ex:

```
for(final int i=0;i<10;i++){

System.out.println(i);

}
```

Status:Compilation Error.

Reason:In General loops it is not suggestible to declare loop variables as final,because,we are able to perform increment and decrement operations over loop variables.

Ex:

```
for(;;)
```

Status:Compilation Error

Ex:

```
for(;;);
```

Status:No Compilation Error

Ex:

```
for(;;)
    ;
```

Status:No Compilation Error

Ex:

```
for(;;){
}
```

Status:No Compilation Error

Reason:In for loop,curly braces are optional in the case of single statement body.

In the case of single statement body we must provide either curly braces or ; to terminate the loop.

In general in Java applications,we are able to utilize for loop when we aware the no.of iterations before writing for loop.

Ex:

To retrieve elements from an array we are able to use "for" loop because we are able to identify the size of the array before writing for loop.

Note:Size of the array is equals to the no of iterations which are required to retrieve elements from array.

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

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

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

}
```

If we use the above for loop to retrieve elements from an array and from collection objects then we are able to get the following problems.

a)We have to manage a loop variable explicitly.

b)We have to execute conditional expression at each and every iteration.

c)We have to perform either increment or decrement operations over loop variables explicitly.

d)We have to retrieve the elements on the basis of array index values,it may provide ArrayIndexOutOfBoundsException when we mishandle with index value.

The above problems may decrease application performance while iterating values from an array or from collection.

In the above context,to improve the application performance,JDK5.0 version has provided a new version of for loop called as "for-Each" loop.

Syntax:

```
for(Array_Data_Type var:Array_Ref_Var){

---body--------------

}
```

This for-each loop will iterate values from the specified array and assign values to the specified variable at each and every iteration.

Ex:

```
for(int x : a){

System.out.println(x);

}


class Test{
```

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

String[] str={"AAA","BBB","CCC","DDD","EEE"};

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

System.out.println(str[i]);

}

System.out.println();

for(String s: str){

System.out.println(s);

}}}
```

2.while loop:

- In Java applications,we are able to utillize while loop when we are not aware the no. of iterations before writing loop.

Syntax:

```
while(conditional_Expression){

----body--------

}
```

```
class Test{

public static void main(String args[]){

int i=0;

while(i<10){
```

```
System.out.println(i);
i=i+1;
}}}
```

Ex:

```
int i=0;
while(){
System.out.println(i);
i=i+1;
}
```

Status:Compilation Error

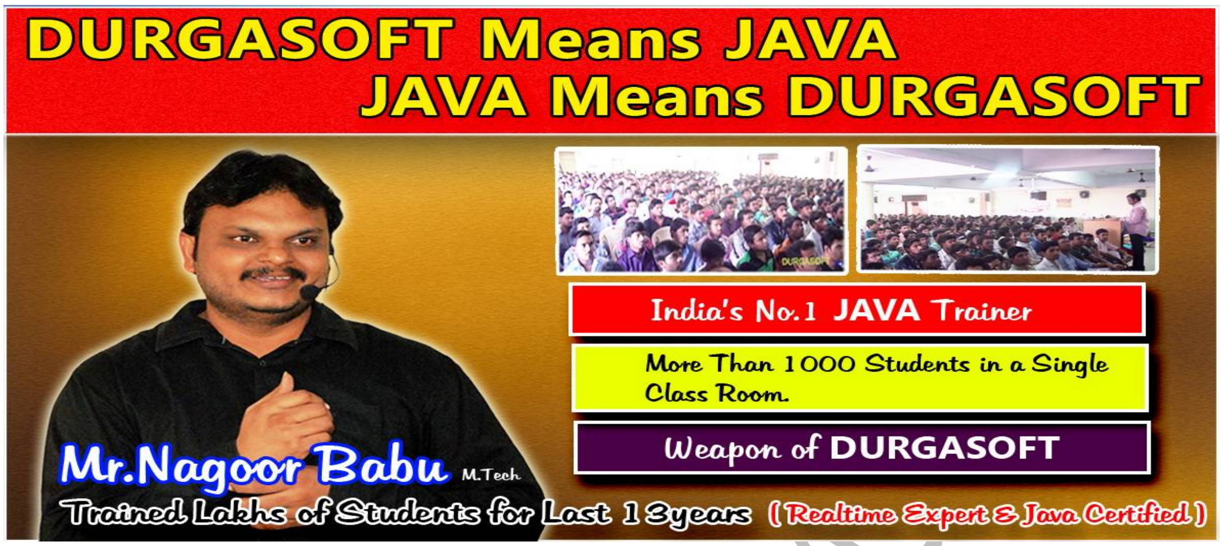
Reason:In the case of while loop,conditional expression is mandatory.

Ex:

```
int i=0;
System.out.println("Before Loop");
while(i<10){
System.out.println("Inside loop");
}
System.out.println("After loop");
```

Status:No Compilation Error

O/P:Infinite loop


DURGASOFT Means JAVA
JAVA Means DURGASOFT
India's No.1 JAVA Trainer
More Than 1000 Students in a Single Class Room.
Weapon of DURGASOFT
Mr.Nagoor Babu M.Tech
Trained Lakhs of Students for Last 13years (Realtime Expert & Java Certified)

Ex:

```
int i=0;
System.out.println("Before for Loop");
while(true){
System.out.println("Inside loop");
}
System.out.println("After loop");
```

Status:Compilation Error

do-while loop:

Q)What are the differences between while loop and do-while loop?

Ans:1.In the case of while loop there is no gaurnatee whether the loop body is executed minimum one time.

In the case of do-while loop body will be executed minimum one time.

2.In the case of while loop,first condition will be executed then body will be executed.

In the case of do-while loop,first body will be executed then condition will be checked.

3.In the case of while loop condition will be checked to perform present iteration.

In the case of do-while loop condition will be checked to perform next iteration.

```
Syntax:
do{
---body--
}while(condition);
Ex:
int i=0;
do{
System.out.println(i);
i=i+1;
}
while(i<10);

Ex:
int i=0;
do{
System.out.println(i);
i=i+1;
}
while();
Status:Compilation Error
Reason:Conditional Expression is mandatory
```

Transfer Statements:

These are Java Statements,which can be used to bypass the flow of execution from one instruction to another instruction.

There are 3 transfer statements in java

1.break

2.continue

3.return

1.break:

- This transfer statement can be used to skip all the remaining instructions at present iteration,it will skip all the remaining iterations and it will bypass flow of execution to outside of the loop

Ex:

```
for(int i=0;i<10;i++){
if(i==5){
break;
}
System.out.println(i);
}
```

Ex:

```
for(int i=0;i<10;i++){
if(i==5){
System.out.println("Before break");
break;
System.out.println("After break");
}
System.out.println(i);

}
```

Status:Compilation Error(unreachable statement)

Reason:If we provide any statement immediately after "break" statement then that statement is "unreachable statement".

Ex:

```
for(int i=0;i<10;i++){

for(int j=0;j<10;j++){

System.out.prinltn(i+"     "+j);

if(j==5){

break;

}

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

}}
```

- In Java,if we provide break statement in an inner loop then that "break" statement will give effect to inner loop only,it will not give effect to outer loop.
- In the above context,if we want to give effect to outer loop by keeping "break" statement in inner loop then we must go for "labelled break" statement break label;

  where "label" must be marked with outer loop.

Ex:

```
for(int i=0;i<10;i++){
for(int j=0;j<10;j++){
System.out.priniltn(i+"    "+j);
if(j==5){
break l1;
}
System.out.println(i+"    "+j);
}}
```

continue:

This transfer statement can be used to skip the remaining instructions in the present iteration and continue with the next iteration.

Ex:

```
for(int i=0;i<10;i++){
if(i==5){
continue;
}}
System.out.println(i);
```

Ex:

```
for(int i=0;i<10;i++){
if(i==5){
```

```
System.out.println("before continue");
continue;
System.out.println("After continue");
}}
System.out.println(i);
```

Status:Compilation Error

Reason:If we provide any statement immediately after continue statement then that statement is called as unreachable statement where compiler will rise an error.

Ex:

```
for(int i=0;i<10;i++){
for(int j=0;j<10;j++){
if(j==5){
continue;
}
System.out.println(i+"    "+j);
}}
```

If we provide continue statement inside an inner loop then continue statement will give effect to inner loop only,it will not give effect to outer loop.

Ex:

```
for(int i=0;i<10;i++){
```

```
for(int j=0;j<10;j++){

System.out.prinltn(i+"     "+j);

if(j==5){

continue l1;

}

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

}}
```

- In the above context,if we want to give continue statement effect to outer loop by keeping continue statement in inner loop then we have to use "labelled continue" statement continue label;

  where "label" must be marked with outer loop.

DURGASOFT Means JAVA
JAVA Means DURGASOFT
India's No.1 JAVA Trainer
More Than 1000 Students in a Single Class Room.
Weapon of DURGASOFT
Mr.Nagoor Babu M.Tech
Trained Lakhs of Students for Last 13years (Realtime Expert & Java Certified)