UNIT-II

Operators

* Java provides a rich operator environment.
  * the operators are divided into four groups → Arithmatic, bitwise, Relational, Logical.

Arithmatic operators :- are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmatic operators:

operator	Result
+	Addition
−	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
−−	Decrement
+z	Addition Assignment
−z	Subtraction Assignment.
*z	Multiplication Assignment.
/z	Division Assignment.
%z	Modulus Assignment.

* the operands of the arithmatic operators must be of a numeric type. (cannot be used on booleans but on char).

* Simple example program to demonstrate arithmatic operators

```
Class BasicMath {
```

```
public static void main (String args[]){
System.out.println("Integer Arithmetic");
    int a = 1+1;
    int b = a*3;
    int c = b/4;
    int d = c-a;
    int 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);
System.out.println("\n Floating point Arithmetic");
    double da = 1+1;
    double db = da*3;
    double dc = db/4;
    double dd = dc-a;
    double de = -dd;
System.out.println("da=" +da);
System.out.println("db=" +db);
System.out.println("dc=" +dc);
System.out.println("dd=" +dd);
System.out.println("de=" +de); } }
```

o/p:- Integer Arithmetic

a = 2,
b = 6,
c = 1
d = -1
e = 1

Floating point Arithmetic

da = 2.0
db = 6.0
dc = 1.5
dd = -0.5
de = 0.5.

The Modulus operator.

* The Modulus operator %, returns the remainder of the division operation.
* It can be applied to integer & floating-point numbers.
* Sample program to Demonstrate Modulus % operator.

```
class Modulus {
public static void main (String args[]) {
    int x = 42,
    double y = 42.25.
System. out. println ("x mod 10 =" + x %. 10),.
System. out. println ("y mod 10 =" + y %. 10),.
}
}
```

o/p:-

x mod 10 = 2
y mod 10 = 2.25

```
      4
x%.10 = 10) 42
           40
           ---
           42
           40
           ---
           2

      4
y%.10 = 10) 4225
           40.00
           -----
           2.25
```

Arithmetic Compound Assignment operators.

* Java provides special operators that can be used to Combine an arithmetic operation with an assignment.

Eg:- a = a+4;

It can also be written as a+=4;.

* The operator "+=" is called compound assignment operator. Another eg:-

a = a%2 which can be written as

a%=2;.

General form:- Any expression of form.

Var = Var op expression; can be written as

Var op = expression;

Benefits of compound assignment operators

1) Bit of typing is saved because they are "shorthand"

2) They are implemented more efficiently by the Java run-time system than are their equivalent long forms.

Example Program:-

```
class opequals{
public static void main(String args[])
{
    int a=1,b=2,c=3;
    a+=5;
    b*=4;
    c+=a*b;
    c%=6;
System.out.println("a = "+a);
```

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

c=51
b=3
8 6 0 3
6)51 18
48 5
3
=

o/p:- a=6
b=8
c=3

Increment & Decrement.

* ++ & -- are increment & decrement operators respectively.
* Increment operator increases its operand by one & decrement operator decreases its operand by one.

For eg:- x=x+1; can be rewritten as x++;

Similarly, x=x-1; can be written as x--;

Note:- ▷ In the prefix form, the operand is incremented & decremented before the value is obtained for use in the expression.
▷ In the postfix form, the previous value is obtained for use in the expression, & then the operand is modified.

For eg:- x=42;
y=++x;

In the case, y=43, because 'x' is incremented first.
x=43.

Thus the above code is a shorthand for:

x=42;
y=x+1;

In the next case,

x=42;
y=x++;

value of x is 43 & y is 42.

Thus, it can be interpreted as.

y=x;
x=x+1;

Example program

```
Class incdec {
  Public static void main (String args[]) {
    int a=1, b=2, c, d;
      c = ++b;
      d = a++;
      ++c;
    System.out.println ("a = " +a);
    System.out.println ("b = " +b);
    System.out.println ("c = " +c);
    System.out.println ("d = " +d);
  }
}
```

O/P:- a = 2
b = 3
c = 4
d = 1

The Bitwise Operators :- acts upon the individual bits of their operands.

Operator	Result
~	Bitwise unary NOT.
&	Bitwise AND.
|	Bitwise OR.
^	Bitwise exclusive OR.
>>	Shift Right
>>>	Shift right zero fill.

`<<` Shift left

`&=` Bitwise AND assignment

`|=` Bitwise OR assignment.

`^=` Bitwise exclusive OR assignment.

`>>=` Shift right assignment.

`>>>=` Shift right zero fill assignment.

`<<=` Shift left assignment.

* Integers are stored as binary numbers. For eg:- 42 is stored as 00101010, where each position represents a power of two, starting with 2^0 at the rightmost bit.

Thus,

00101010
$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$
$2^5 + 2^3 + 2^1$ =
$32 + 8 + 2 = 42$

Note:- Count only the bits that are '1'.

* Java uses encoding known as 2's complement i.e. negative numbers are represented by inverting all of the bits in a value, then adding 1 to the result.

For eg:- $-42 = 11010101 + 1$

11010110

The Bitwise logical operators

* The bitwise logical operators are &, |, ^, & ~.

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

The following program demonstrates the bitwise logical operators.

```
Class Bitlogic{
public static void main (String args[])
{
    String binary [] = { "0000", "0001", "0010", "0011", "0100",
    "0101", "0110", "0111", "1000", "1010", "1011", "1100", "1101",
    "1110", "1111" };
    int a = 3;
    int b = 6;
    int c = a|b;
    int d = a&b;  int e = a^b;
    int f = (~a &b) | (a &~b);
    int g = ~a & 0x0f;
    System.out.println("a = " + binary[a]);
    System.out.println("b = " + binary[b]);
    System.out.println("a|b = " + binary[c]);
    System.out.println("a&b = " + binary[d]);
    System.out.println("a^b = " + binary[e]);
    System.out.println("~a&b | a&~b = " + binary[f]);
    System.out.println("~a = " + binary[g]);
}
}
```

o/p :- a = 0011, b = 0110. c = a|b

c = 0111

```
  0011
| 0110
------
  0111
```

d = a|b = 0011
0110
d = 0010

e = a^b = 0011
0110
e = 0101.

f = ~a&b | a&~b

~a = 1100

1100 0011
& 0110 & 1001
0100 | 0001

0100
0001
f = 0101

The left shift :- The left shift operator, <<, shifts all of the bits in a value to the left a specified number of times. It has this general form:

value << num.
↓ specifies the number of positions to left-shift the value.

Note:- 1) When a left shift is applied to an int operand, bits are lost once they are shifted past bit position 31.

2) Java's automatic type promotions produce unexpected results when you are shifting byte & short values. byte & short values are promoted to int when an expression is evaluated. Thus, the bits are lost once they are shifted past bit position 31.

Example for Left Shifting a byte value.

```
class ByteShift{
public static void main(String args[]) {
    byte a=64, b;
    int i;
    i = a<<2;
    b = (byte)(a<<2);
```

```
System.out.println("Original value of a: "+a);
System.out.println("i & b: "+i+" "+b);
    }
}
```

O/P:- Original value of a: 64
i & b: 256 0

{ Note:- alternate is multiplying by 2
i.e. 64×2
=256 }

64 = 0100 0000
↙ 1000 0000
↙ 1 0000 0000 = 256 = i
b = 0.

The Right Shift:- The right shift operator shifts all of the bits in a value to the right a specified number of times. Its general form is:

value >> num.

Eg:- int a = 32;
a = a>>2; a = 8

1000 0000
↘ 0100 000
↘ 0010 00 = 8.

Bitwise Operator Compound Assignments.

* All of the binary bitwise operators have a compound form similar to that of the algebraic operators, which combines the assignment with the bitwise operation.

For eg:- a = a >> 4;
a >>= 4;

Also, a = a|b;
a|= b;

Relational operators

* The relational operators determine the relationship that operand has to the other. Specifically, they determine equality & ordering. The relational operators are shown here.

operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to.

Note:- The outcome of these operations is a boolean value. The relational operators are most frequently used in the expressions that control the if statements & the various loop statements.

* The result produced by a relational operator is a boolean value.

```
int a = 4, b = 1;
boolean c = a < b;
```

The result "false" is stored in 'c'.

Boolean Logical operators

Operator	Result
&	Logical AND
|	Logical OR
^	" XOR
||	Short circuit OR

&&	Short-circuit AND.
!	Logical unary NOT.
&=	AND assignment.
|=	OR assignment.
^=	XOR assignment.
==	Equal to
!=	Not equal to
?:	Ternary if-then-else.

```
class Boollogic{
public static void main(String args[]){
    boolean a = true;
    boolean b = false;
    boolean c = a|b;
    boolean d = a&b;
    boolean e = a^b;
    boolean y = (!a&b)|(a&!b);
    boolean g = !a;
System.out.println("a="+a);
System.out.println("b="+b);
System.out.println("a|b="+c);
System.out.println("a&b="+d);
System.out.println("a^b="+e);
}
}
```

O/P:- a = true, b = false, a|b = true, a&b = false

a^b = true, a&b | a&!b = true ! a = false.

Short-Circuit Logical operators.

* Java provides two interesting Boolean operators Short-Circuit OR || & Short-Circuit AND &&.

* The OR operator results in true when A is true, no matter what B is. Similarly, the AND operator results in false when A is false, no matter what B is. If you use the || and && forms, rather than the | and & forms of these operators, Java will not bother to evaluate the right-hand operand when the outcome of the expression can be determined by the left operand alone. This is very useful when the right-hand operand depends on the value of the left one in order to function properly.

For eg:- if (denom != 0 && num/denom > 10)

Since the short-circuit form of AND (&&) is used, there is no risk of causing a run-time exception when denom is Zero.

The Assignment Operator (=).

* The assignment operator is the single equal sign, =. The

* the general form is:

var = expression;

↑ ——— ↑

compatability has to be checked.

Eg:- int x, y, z;

x = y = 100;

z = 10;

The ? operator

* Java includes a special ternary (three-way) operator that can replace certain types of if-then-else statements. This is the operator ?. The ? has general form:

expression1 ? expression2 : expression3.

*If expression1 is true, then expression2 is evaluated, otherwise expression3 is evaluated.

Eg:- ratio = denom = 0 ? 0 : num/denom.

```
class Ternary {
public static void main(String args[]){
    int i, k;
    i = 10;
    k = i < 0 ? -i : i;
System.out.println("Absolute value of ");
System.out.println(" i + " is " + k);
    i = -10;  k = i < 0 ? -i : i;
System.out.println(" Absolute value of ");
System.out.println(" i + " is " + k);
}
}
```

O/P:- Absolute value of 10 is 10
Absolute value of -10 is 10.

Operator precedence

* Parentheses, square brackets & dot operator. Technically these are separators but they can also be used as operators.

Highest

()	[ ]	.	
++	--	~	!
*	/	%	
+	-		
>>	>>>	<<	
>	>=	<	<=
==	!=		
&			
^			
|			
&&			
||			
?:			
=	op=		

Lowest

Chapter-2 (Module-II)

* Programming language uses control statements to cause the flow of execution to advance & branch based on changes to the state of a program.
* JAVA's program control statements can be put into the following categories: selection, iteration & Jump.
* Selection statements allow your program to choose different paths of execution based upon the outcome of an expression or the state of a variable.

* Iteration statements enable program execution to repeat one or more statements (i.e. form of loops).
* Jump statements allow your programs to run in Non-linear fashion.

Java's Selection statements.

- Java supports two selection statements: if & switch.

The if statement

- The if statement executes a block of code only if the specified expression is true.
- If the value is false, then the if block is skipped & execution continues with rest of the program.

Note:- Condition Expression must be a Boolean expression.

Syntax:-

```
if (<conditional expression>){
    <statements>
}
```

Eg:-

```
public class Example {
public static void main(String args[]){
int a=10, b=20;
if (a>b)
System.out.println("a>b");
if (a<b)
```

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

}

}

The If else statement :- used to route program execution through two different paths.

Syntax:-

If (<conditional expression>){ // If condition is true statement 1 executes

Statement 1;

else Statement 2; else Statement 2.

Where, each statement may be a single statement or a compound statement enclosed in curly braces (that is, a block).

Eg:- ~~Pub~~ Class Example {

public static void main(String args[])

{

Int a = 10, b = 20;

If (a>b)

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

else

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

}

}

Nested Ifs:-

- A Nested If is an If statement that is the target of another If or else.
- When you Nest Ifs, the main thing to remember is that an else statement always refers to the nearest If

statement, that is within the same block as the else & that is not already associated with an else.

Eg:
```
if (i==10) {
    if (j<20) a=b;
    if (k>100) c=d; //
    else a=c;       //
}
else a=d;
```
associated here ← associated

The if-else-if ladder.

* A common programming construct that is based upon a sequence of nested ifs is the if-else-if ladder.

Syntax:
```
if (condition)
    statement;
else if (condition)
    statement;
else if (condition)
    statement;
    .
    .
    .
else
    statement;
```

* The if statements are executed from the top to down.
* As soon as one of the conditions controlling the if is true, the statement associated with that if is executed & the rest of the ladder is bypassed.
* If none of the condition is true, then the final else

statement will be executed.

Eg:- class IfElse {
    public static void main (String args[]) {
        int month = 4; // April.
        String Season;
        if (month == 12 || month == 1 || month == 2)
            Season = "Winter";
        else if (month == 3 || month == 4 || month == 5)
            Season = "Spring";
        else if (month == 6 || month == 7 || month == 8)
            Season = "Summer";
        else if (month == 9 || month == 10 || month == 11)
            Season = "Autumn";
        else
            Season = "Bogus month";
        System.out.println ("April is in the " + Season + ".");
    }
}

o/p:- April is in the Spring.

The Switch Statement.

* The switch case statement is a multi-way branch with several choices. A switch is easier to implement than a series of if/else statements.

Structure of Switch:-

* The switch statement begins with a keyboard, followed by an expression that equates to a no long integral value.
* Following the controlling expression is a code block that

Contains zero of more labelled cases.

* Each label must equate to an Integer constant & each must be unique.

Working of Switch case:-

* When the switch statement executes, it compares the value of the controlling expression to the values of each case label.
* The program will select the value of the case label that equals the value of the controlling expression & branch down that path to the end of the code Block.
* If none of the case label values match, then none of the codes within the switch code block will be executed. Java uses a default label to use in cases where there are no matches.

Syntax:-

```
Switch(<non-long Integral expression>){
    Case label 1 : <statement 1>; break;
    case label 2: <statement 2>; break.
    .
    .
    case labeln: <statementn>; break;
    default: <statement>
}
```

Eg:-

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

```
for (int i=0; i<6; i++)
  switch (i) {
   case 0:
System.out.println ("i is zero");
    break;
   case 1:
System.out.println ("i is one");
    break;
   case 2:
System.out.println ("i is two");
    break;
default:
    System.out.println ("i is greater than 2.");
}
}
```

O/P:- i is zero.
i is one.
i is two.
i is greater than 2.

* The break statement is optional. If you omit break, execution will continue on into the next case.
* It is sometimes desirable to have multiple cases without break statements between them.

Eg:-
```
class MissingBreak {
 public static void main (String args[]) {
  for (int i=0; i<12; i++)
      switch (i) {
      case 0:
      case 1:
```

```
        case 2:
        case 3:
        case 4:
System.out.println("i is less than 5");
            break;
        case 5:
        case 6:
        case 7:
        case 8:
        case 9:
System.out.println("i is less than 10");
            break;
        default:
System.out.println("i is 10 or more");
    }
}
```

O/P:-

```
i is less than 5
i is less than 5
i is less than 5
i is less than 5
i is less than 5
i is less than 10
i is less than 10.
i is less than 10.
i is less than 10.
i is less than 10.
i is 10 or more
i is 10 or more.
```

Nested Switch Statements.

* you can use a switch as part of the statement sequence of an outer switch. This is called a nested switch.
* Since a switch statement defines its own block, no conflicts arise b/w the case constants in the inner switch & those in the outer switch.

For eg:- switch (count) {

```
    case 1:
      switch (target) { // nested switch
       case 0:
System.out.println ("target is zero");
        break;
       case 1: // no conflicts with outer switch
System.out.println ("target is one");
        break;
      }
      break;
    case 2: //- - -
```

*** The three important features of the switch statement are:-

1) The switch differs from the if in that switch can only test for equality, whereas if can evaluate any type of Boolean expression i.e. the switch looks only for a match between the value of the expression & one of its case constants.
2) No two case constants in the same switch can have identical values. Of course, a switch statement & an enclosing outer switch can have case constants in common.
3) A switch statement is usually more efficient than a

set of nested ifs.

Iteration Statements:- (for, while, do-while).

The while loop:-

* The while statement is a looping construct control statement that executes a block of code while a condition is true.
* you can either have a single statement or a block of code within the while loop.
* The loop will never be executed if the testing expression evaluates to false.
* the loop condition must be a boolean expression.

Syntax:- while (< loop condition>) {

< Statements >

}

Eg:- class While {

public static void main (String args[])

{ int n=10;

while (n>0) {

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

n--;

} } }

O/P:- tick 10

tick 9

tick 8

Eck 1.

Note:- The body of the while (or any other of Java's loops) can be empty. This is because a null statement (one that consists of only semi-colon) is syntactically valid in Java.

For eg: class NoBody {
    public static void main (String args[])
    {
        int i, j;
        j=200;
        i=100;
// find mid-point between i & j.
        while (++i<--j); // No body in this loop.
        System.out.println ("Midpoint is " + i);
    }
}

O/P:- Midpoint is 150.

do-while.

* the do-while loop is similar to the while loop, except that the test is performed at the end of the loop instead of at the beginning.
* This ensures that the loop will be executed at least once.

Syntax:- do {
    // body of the loop
} while (condition);

Eg:-
```
class Example{
    public static void main(String[] args){
        int count=1;
        System.out.println("printing numbers from 1 to 10");
        do{
            System.out.println(count++);
        }while(count<=10);
    }
}
```

The for loop

* The for loop is a looping construct which can execute a set of instructions a specified number of times. It's a counter controlled loop.

Syntax:-

```
for(initialization; condition; iteration)
{
// body [Executed only once]
}
```

[Expression]

*** No need of curly braces if there is only one statement

Eg:-
```
class Example{
    public static void main(String args[]){
        int count;
        System.out.println("printing numbers from 1 to 10");
        for(count=1; count<=10; count++)
        {
            System.out.println(count);
        }
    }
}
```

Declaring Loop control variables Inside the for loop.

* Often the variable that controls the for loop is only needed for the purposes of the loop & is not used elsewhere.
* When this is the case, it is possible to declare the variable inside the initialization portion of the for.

Eg: class FORTest{

public static void main (String args[]) {

// here, n is declared inside the for loop.

for (int n=10; n>0; n--)

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

}

}

Note:- When you declare a variable inside a for loop, the scope of that variable ends when the for loop does.

Using the Comma

* There will be times when you will want to include more than one statement in the initialization & iteration portions of the for loop.

class Sample {

public static void main (String args[])

{

int a,b;

for(a=1,b=4; a<b; a++,b--)

{

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

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

o/p:- a=1
b=4
a=2
b=3

Some for loop variations.

* The for loop supports a number of variations that increases its power & applicability.
* The reason it is so flexible is that its three parts
  - the initialization, the conditional test & the iteration,
  - do not need to be used for only those purposes but can be used for any purpose of your desire.

1) one of the most common variations involves the conditional expression. Specially, this expression does not need to test the loop control variable against some target value.

* The condition controlling the for loop can be any boolean expression.

```
boolean done = false;
for (int i=1; !done; i++) {
    if (Interrupted())
        done = true;
}
```

In this example, the for loop continues to run until the boolean variable done is set to true.

2) Here is another for loop variation. Either the initialization or the iteration expression or both may be absent, as shown in the example program.

```
Class ForVar {
public static void main(String args[]){
    int i;
boolean done = false;
    i = 0;
    for( ; !done;) {
    System.out.println("i is " + i);
        if(i==10) done = true;
        i++;
    }
}
```

* Here, the initialization & iteration expressions have been moved out of the for.

3) One more for loop variation exists, where you can intentionally create an infinite loop (a loop that never terminates). It can be done by leaving all the three parts of the for loop empty.

for eg:

```
for(;;) {
    //---
}   // It will run for ever.
```

The For-Each version of the for loop.

* Beginning with JDK5, a second form of for was defined that implements a "for-each" style loop.

The general form is:

for (type itr-var : collection) statement block.

↓

[elements that are used for iterations ex $i < n$

↑ itr. ↑ collection

* The collection being cycled through is specified as collection. Eg:- array.

* Working:-

- With each iteration of the loop, the next element in the collection is retrieved & stored in itr-var.
- The loop repeats until all the elements in the collection have been obtained.
- Because the iteration variable receives values from the collection, type must be the same as the elements stored in the collection.
- Thus, when iterating over arrays, type must be compatible with the base type of the array.

```
Class Foreach{
public static void main(String args[]){
    int nums[] = {1,2,3,4,5,6,7,8,9,10};
        int sum=0;
    for (int x : nums){
        System.out.println("value is:"+x);
```

```
        Sum = Sum + x;
    }
    System.out.println("Summation:" + sum);
}
}
```

o/p:- value is:1
value is:2
:
value is10
Summation:55

Note:- the for-each style for automatically cycles through an array in sequence from the lowest index to the highest.

* For-each for loop iterates until all elements in an array have been examined, it is possible to terminate the loop early by using a break statement.

Prog:-
```
class ForEach2{
public static void main(String args[])
{
int sum=0;
int nums[]={1,2,3,4,5,6,7,8,9,10};
for(int x:nums){
System.out.println("value is:"+x);
    Sum = Sum + x;
if(x==5) break;
}
System.out.println("Summation of first 5 elements:"
                    + Sum);
}
}
```

O/P:- Value @s 1
Value @s 2
Value @s 3
Value @s 4
Value @s 5
Summation of first 5 elements: 15

Note:- The Iterative variable Int 'x' @s read-only & will not have any effect on nums.

Iterating over Multidimensional Arrays.

* The enhanced version of the for also works on Multidimensional arrays.
* Multidimensional arrays consist of arrays of arrays.

For ex:- A two-dimensional array @s an array of one-dim-ensional arrays.

```
Class Foreach3 {
public static void main(String args[])
{
    Int sum=0;
    Int nums[][] = new Int[3][5];
    for(Int i=0; i<3; i++)
        for(Int j=0; j<5; j++)
            nums[i][j] = (i+1)*(j+1);
    for(Int x[] : nums){
        for(Int y : x){
            sum = sum+y;
}}
```

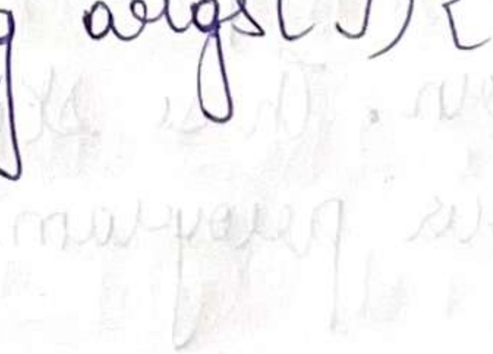
```
        System.out.println("Summation:" + sum);
    }
}
```

O/P:-
```
value of s: 1
value of s: 2
value of s: 3
value of s: 4
value of s: 5
value of s: 2
value of s: 4
value of s: 6
value of s: 8
value of s: 10
value of s: 3
value of s: 6
value of s: 9
value of s: 12
value of s: 15
Summation: 90.
```

program to search group by Element. Nested loops.

```
class Nested{
public static void main (String args[]){
int i, j;
for(i=0; i<10; i++){
  for(j=1; j<10; j++)
   System.out.print("*");
   System.out.println();
}
}
```

O/P:-



Search for a key element.

```
class search {
public static void main (String args[])
{
    int nums[] = {6, 8, 3, 7, 5, 6, 1, 4};
    int val = 5;
    boolean found = false;
    for (int x: nums) {
        if (x == val) {
            found = true;
            break;
    } }
    if (found)
    System.out.println ("value found");
}
```

Jump Statements.

* Java supports three jump statements: break, continue, & return. These statements transfer control to another part of your program.

The break Statement.

- The break statement transfers control out of the enclosing loop (for, while, do & switch statement).
- you use a break statement when you want to jump immediately to the statement following the enclosing control structure.
- A loop can also be provided with a label, & then use

the label in your break statement.

- The label name is optional, & is usually only used when you wish to terminate the outermost loop in a series of nested loops.

Syntax:- break; // unlabelled form
break <label>; // labelled form.

Eg:- class BreakLoop {
public static void main (String args[]) {
for (int i = 0; i < 100; i++) {
if (i == 10) break;
System.out.println("i: " + i);
}
System.out.println ("Loop complete.");
}
}

O/P:- i: 0
i: 1
i: 2
.
.
i: 9
Loop complete.

Eg for Labelled Break

class Break {
public static void main (String args[])
{
boolean t = true;
first: {
second: {
third: {

```
System.out.println("Before the break");
    if(b) break second; // break out of second block.
    System.out.println("This won't execute");
    }
System.out.println("This won't execute");
}
    System.out.println("This is after second block");
    }
}
```

O/P:- Before the Break.
This is after second Block.

The Continue Statement.

* A Continue statement stops the iteration of a loop (while, do & for) & causes execution to resume at the top of the nearest enclosing loop.
* You use a Continue statement when you do not want to execute the remaining statements in the loop, but you do not want to exit the loop itself.
* You can also provide a loop with a label & then use the label in Continue statement.
* The label name is optional, & is usually only used when you wish to return to the outermost loop in a series of nested loops.

Syntax:-

```
Continue;          // the unlabelled form
Continue <label>; // the labelled form.
```

Example for continue

```
class Example {
    public static void main (String args[])
    {
        System.out.println("odd Numbers");
        for (int i=0; i<=10; ++i) {
            if (i%2==0)
                continue;
            System.out.println (i+ "\t");
        }
    }
}
```

O/P:- 01 (If an even number is found, the loop
23 continues without printing a New line).
45
67
89

The return Statement.

* The return statement exits from the current method & control flow returns to where the method was invoked.

Syntax:- The return statement has two forms:

* One that returns a value

```
return val;
```

* One that doesn't return a value

```
return;
```

Example:-

```
class Example {
    public static void main (String args[]) {
        boolean t = true;
```

```
        System.out.println("Before the return");
        if(b) return;  // return to caller.
        System.out.println("Ohes won't execute");
    }
}
```

o/p:- Before the return.
