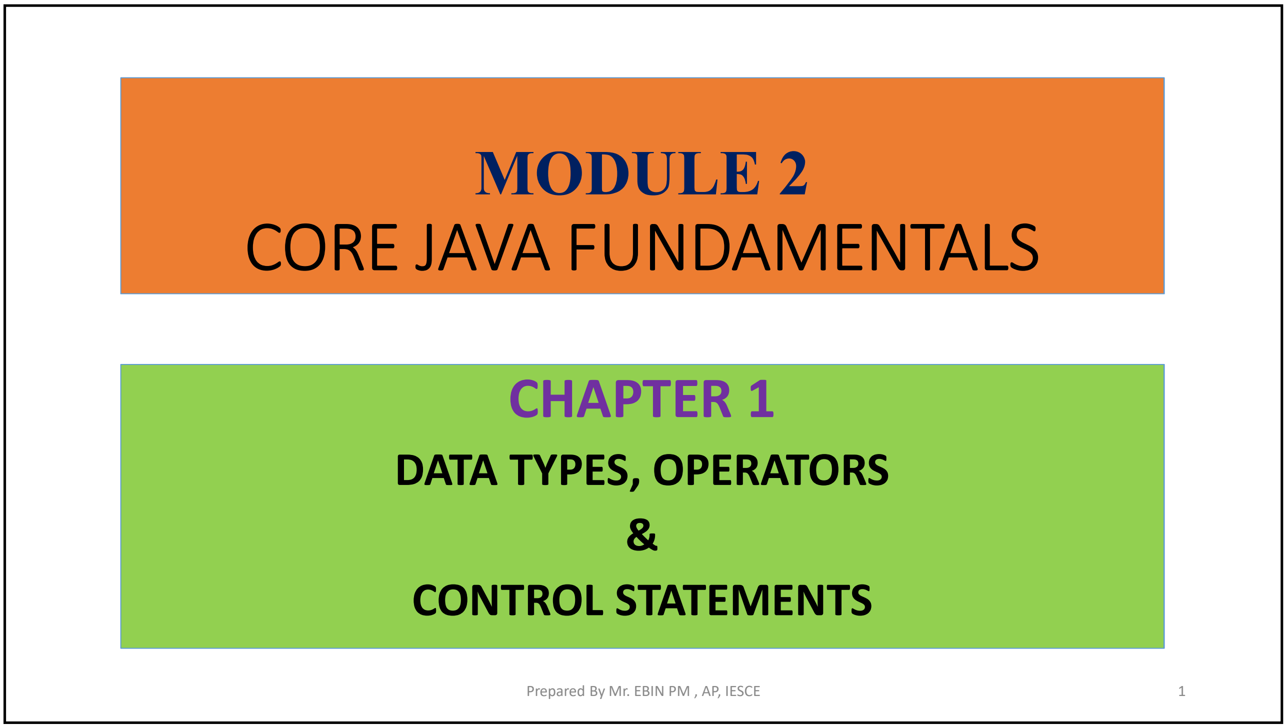


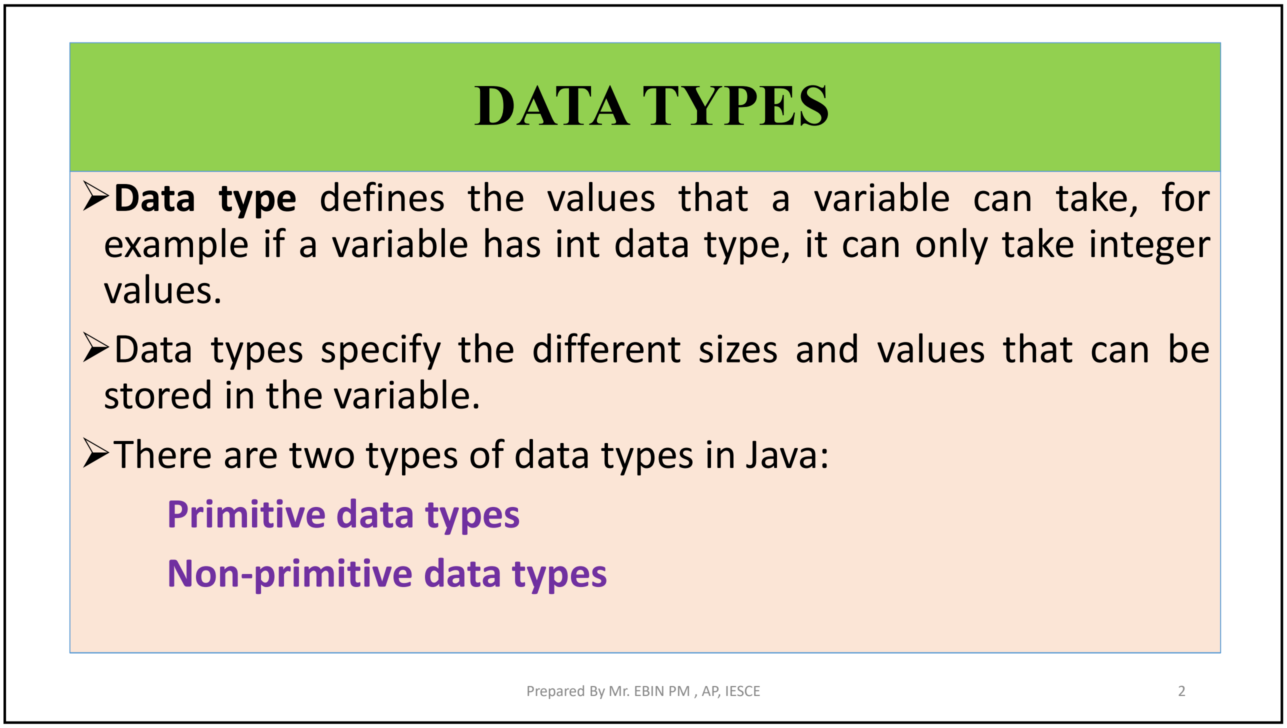
MODULE 2
CORE JAVA FUNDAMENTALS

CHAPTER 1
DATA TYPES, OPERATORS

&

CONTROL STATEMENTS

Prepared By Mr. EBIN PM , AP, IESCE 1



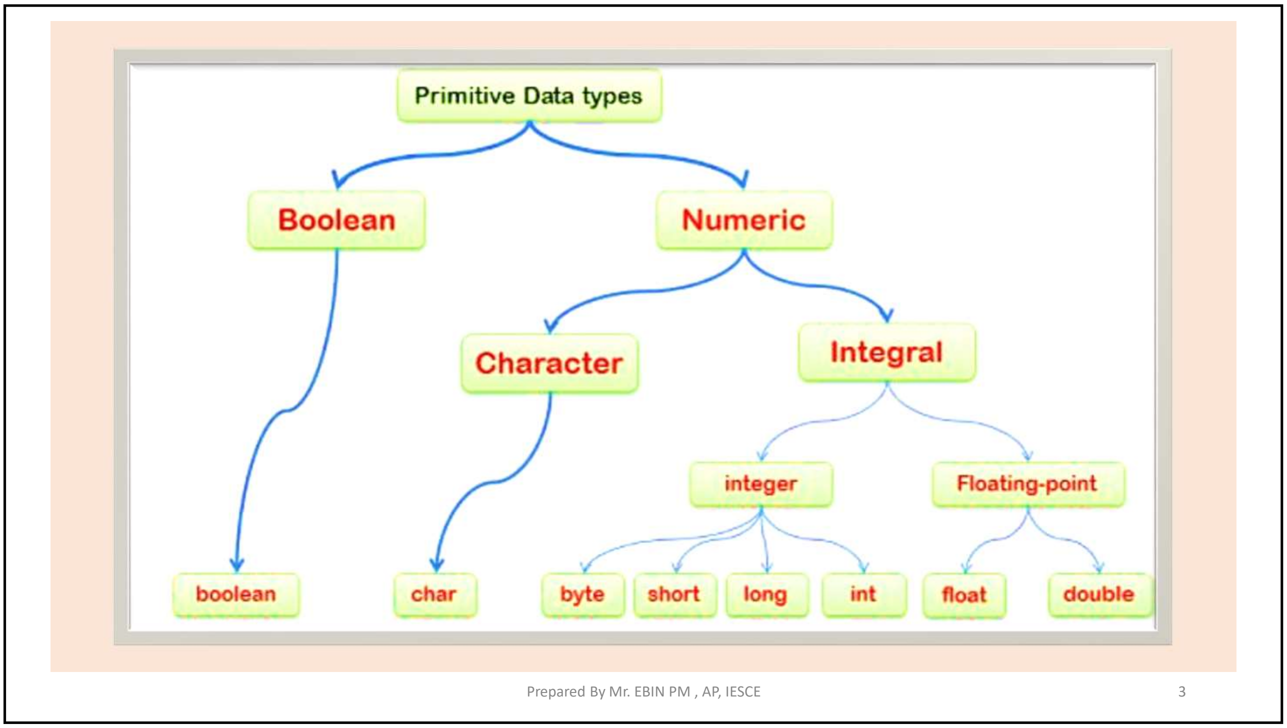
DATA TYPES

- **Data type** defines the values that a variable can take, for example if a variable has int data type, it can only take integer values.
- Data types specify the different sizes and values that can be stored in the variable.
- There are two types of data types in Java:

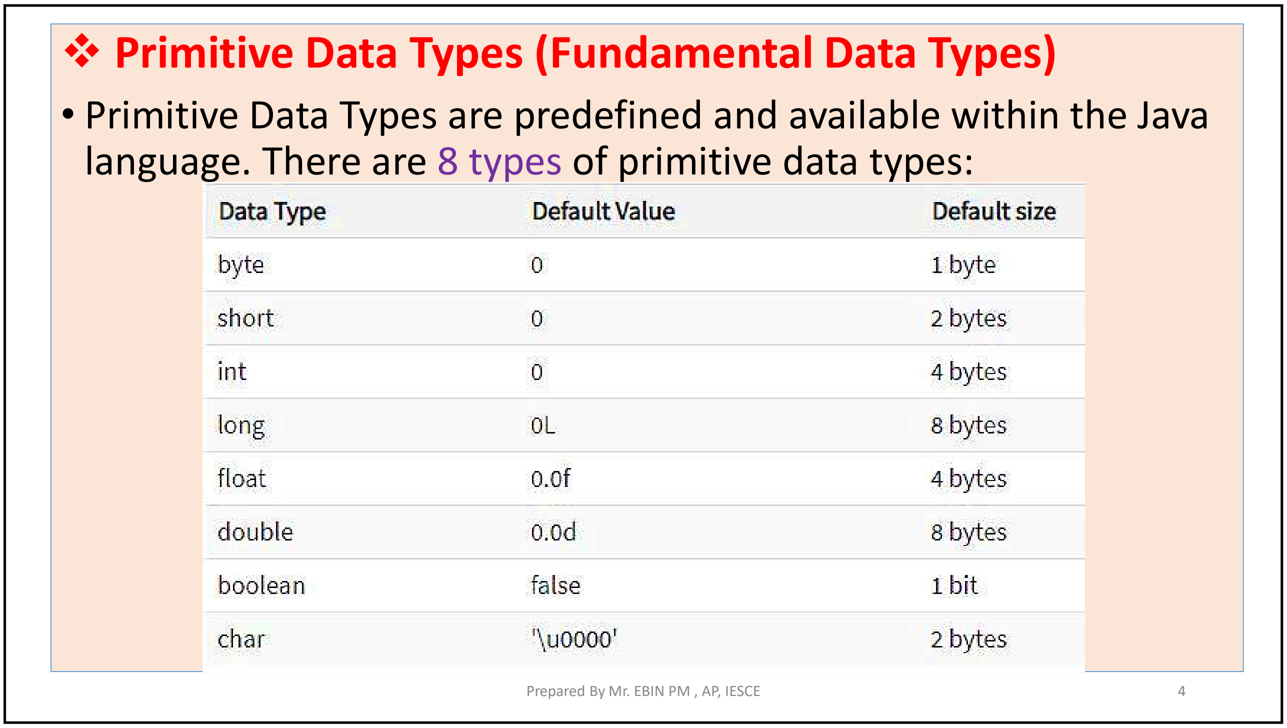
  Primitive data types

  Non-primitive data types

Prepared By Mr. EBIN PM , AP, IESCE 2



Prepared By Mr. EBIN PM , AP, IESCE 3



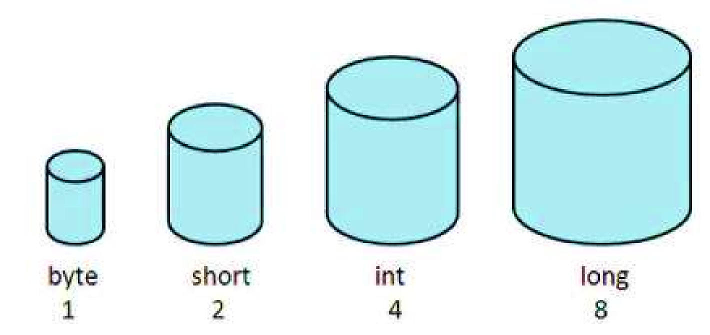
❖ Primitive Data Types (Fundamental Data Types)

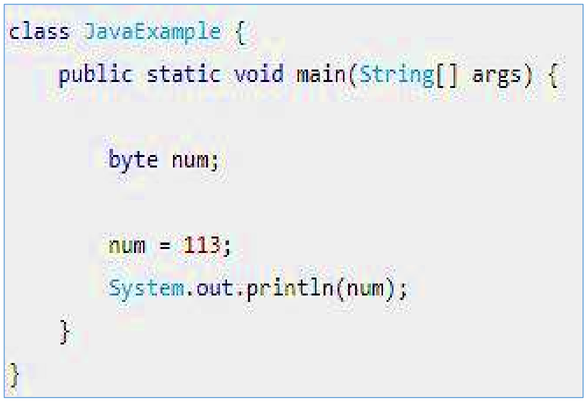
- Primitive Data Types are predefined and available within the Java language. There are 8 types of primitive data types:

Data Type	Default Value	Default size
byte	0	1 byte
short	0	2 bytes
int	0	4 bytes
long	0L	8 bytes
float	0.0f	4 bytes
double	0.0d	8 bytes
boolean	false	1 bit
char	'\u0000'	2 bytes

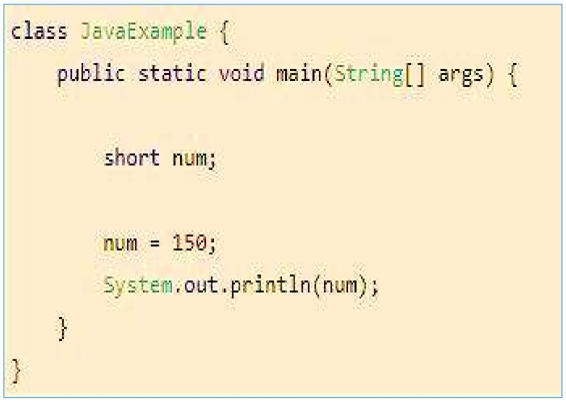
Prepared By Mr. EBIN PM , AP, IESCE 4

- **byte**, **short**, **int** and **long** data types are used for storing whole numbers.
- **float** and **double** are used for fractional numbers.
- **char** is used for storing characters(letters).
- **boolean** data type is used for variables that holds either true or false.





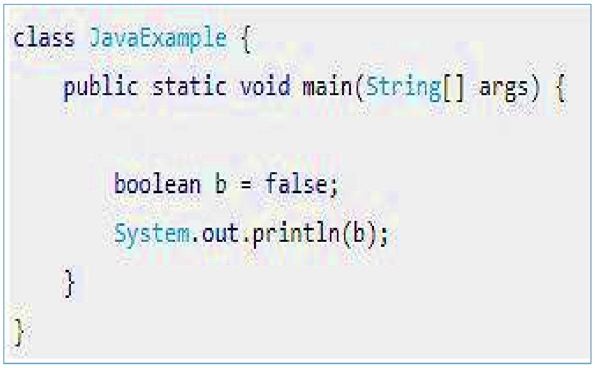
```
class JavaExample {
    public static void main(String[] args) {

        byte num;

        num = 113;
        System.out.println(num);
    }
}
```



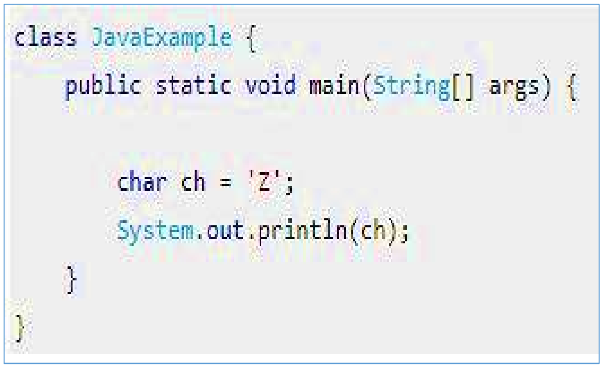
```
class JavaExample {
    public static void main(String[] args) {

        short num;

        num = 150;
        System.out.println(num);
    }
}
```

Output 113

Output 150



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

        boolean b = false;
        System.out.println(b);
    }
}
```



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

        char ch = 'Z';
        System.out.println(ch);
    }
}
```

Output false

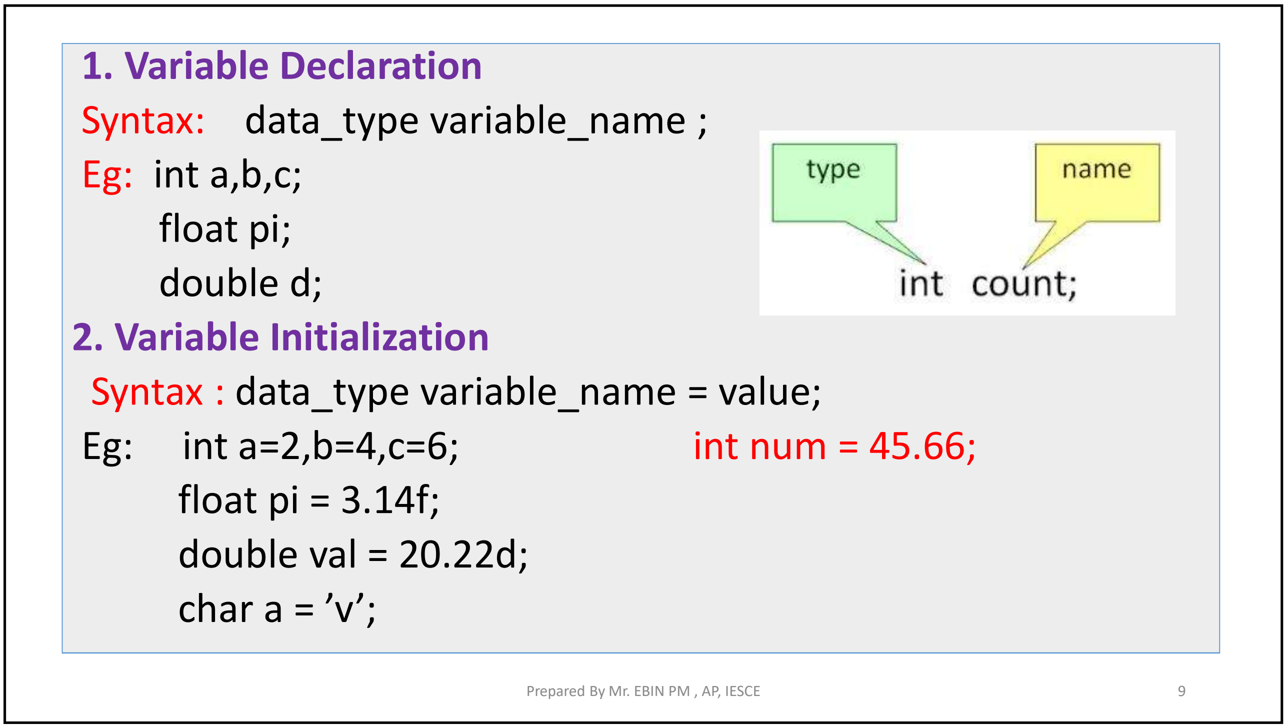
Output Z

Variables In JAVA

- **Variable in Java** is a data container that stores the data values during Java program execution.
- Variable is a memory location name of the data.
- variable="vary + able" that means its value can be changed.
- In order to use a variable in a program we need to perform 2 steps

  1. Variable Declaration

  2. Variable Initialization



1. Variable Declaration

Syntax: data_type variable_name ;

Eg: int a,b,c;

float pi;

double d;

type
name
int count;

2. Variable Initialization

Syntax : data_type variable_name = value;

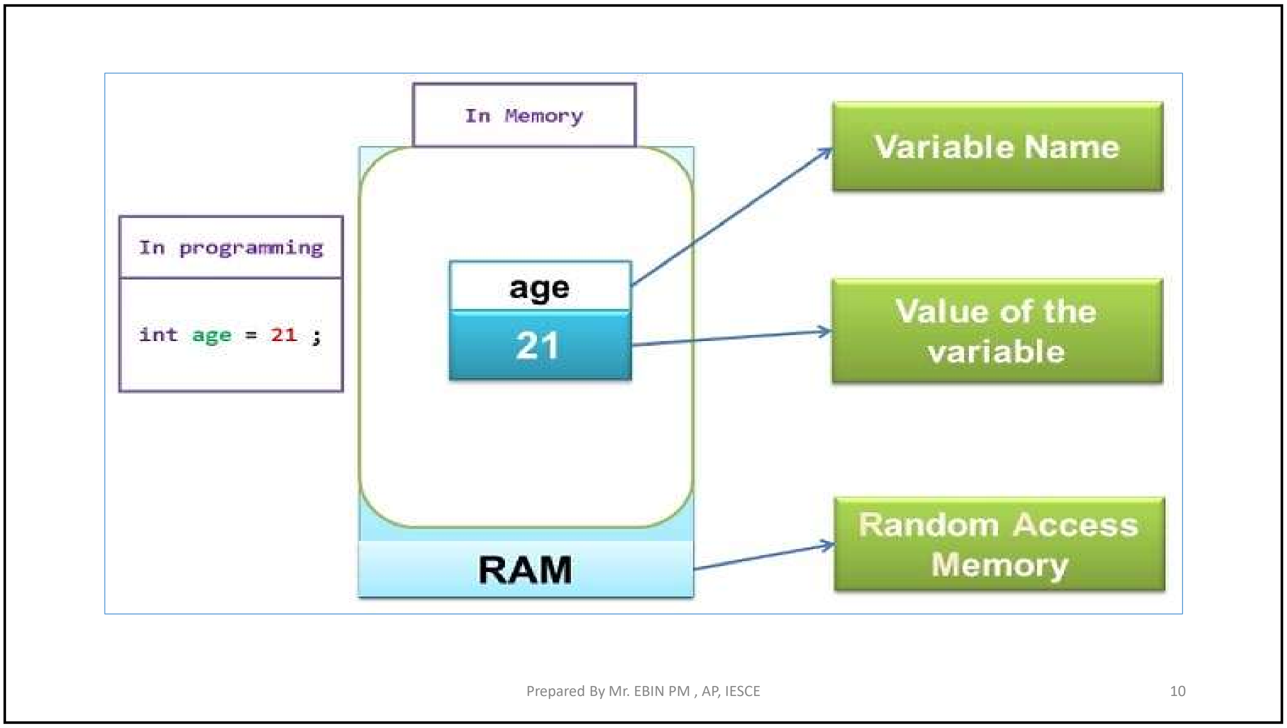
Eg: int a=2,b=4,c=6; int num = 45.66;

float pi = 3.14f;

double val = 20.22d;

char a = ’v’;

Prepared By Mr. EBIN PM , AP, IESCE 9



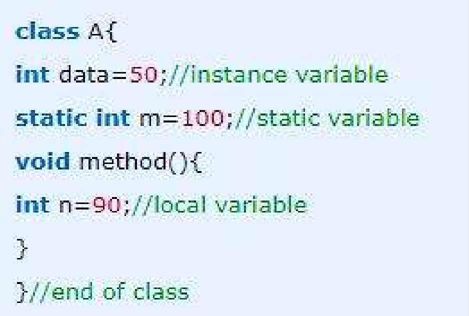
```
int age = 21 ;
```

Prepared By Mr. EBIN PM , AP, IESCE 10

Types of variables

1. **Local variables -** declared inside the method.
2. **Instance Variable -** declared inside the class but outside the method.
3. **Static variable -** declared as with static keyword.

Example:

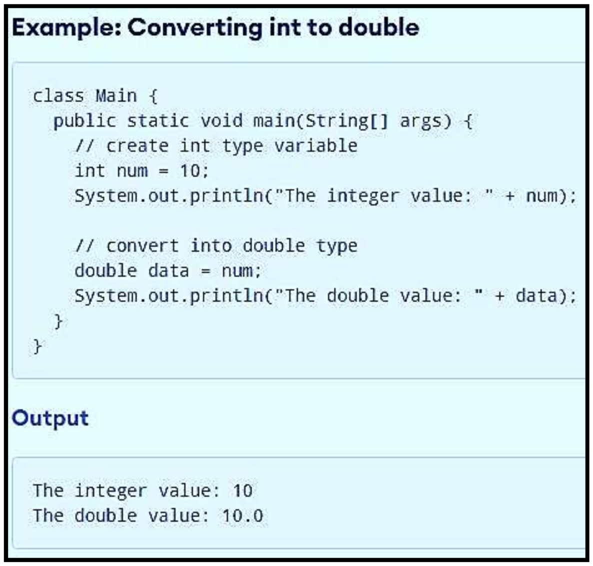


```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

Java Type Casting or Type Conversion

- Type casting is when you assign a value of one primitive data type to another type.
- In Java, there are two types of casting:

1. Widening Casting (automatically) – converting a smaller type to a larger type size (called Type Conversion)
   byte -> short -> char -> int -> long -> float -> double
2. Narrowing Casting (manually) – converting a larger type to a smaller size type (called Type Casting)
   double -> float -> long -> int -> char -> short -> byte

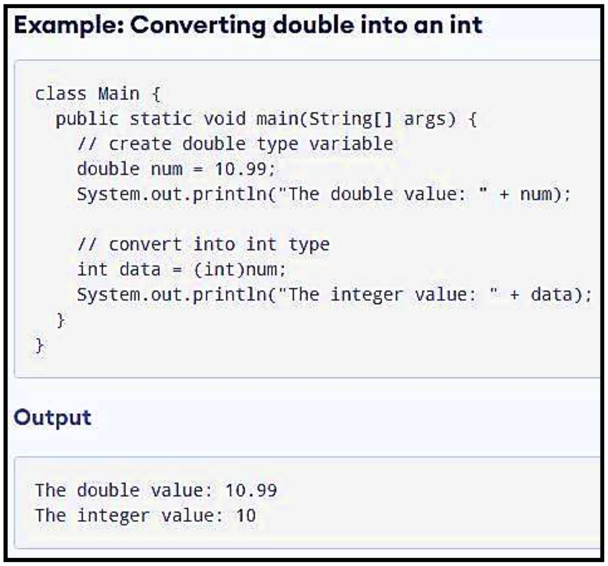


Example: Converting int to double

```
class Main {
  public static void main(String[] args) {
    // create int type variable
    int num = 10;
    System.out.println("The integer value: " + num);

    // convert into double type
    double data = num;
    System.out.println("The double value: " + data);
  }
}
```

Output

```
The integer value: 10
The double value: 10.0
```



Example: Converting double into an int

```
class Main {
  public static void main(String[] args) {
    // create double type variable
    double num = 10.99;
    System.out.println("The double value: " + num);

    // convert into int type
    int data = (int)num;
    System.out.println("The integer value: " + data);
  }
}
```

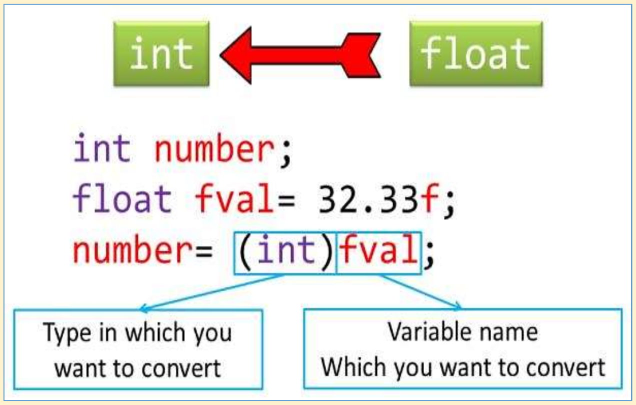
Output

```
The double value: 10.99
The integer value: 10
```

Widening **Narrowing**

❖Truncation

- ➢when a floating-point value is assigned to an integer type: truncation takes place, As you know, integers do not have fractional components
- ➢Thus, when a floating-point value is assigned to an integer type, the fractional component is lost.
- ➢For example, if the value 45.12 is assigned to an integer, the resulting value will simply be 45. The 0.12 will have been truncated.
- ➢No automatic conversions from the numeric types to char or boolean. Also, char and boolean are not compatible with each other.

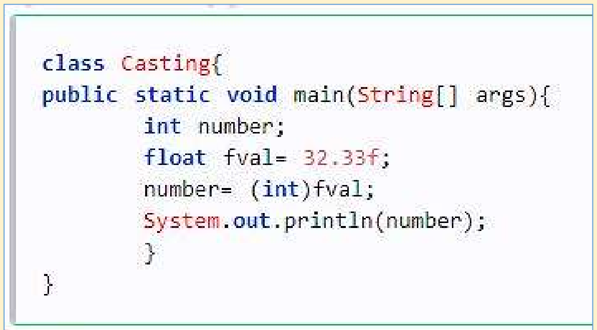


int ← float

```
int number;
float fval= 32.33f;
number= (int)fval;
```

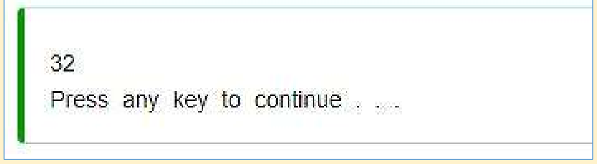
Type in which you want to convert

Variable name Which you want to convert



```
class Casting{
public static void main(String[] args){
        int number;
        float fval= 32.33f;
        number= (int)fval;
        System.out.println(number);
        }
}
```

Output:

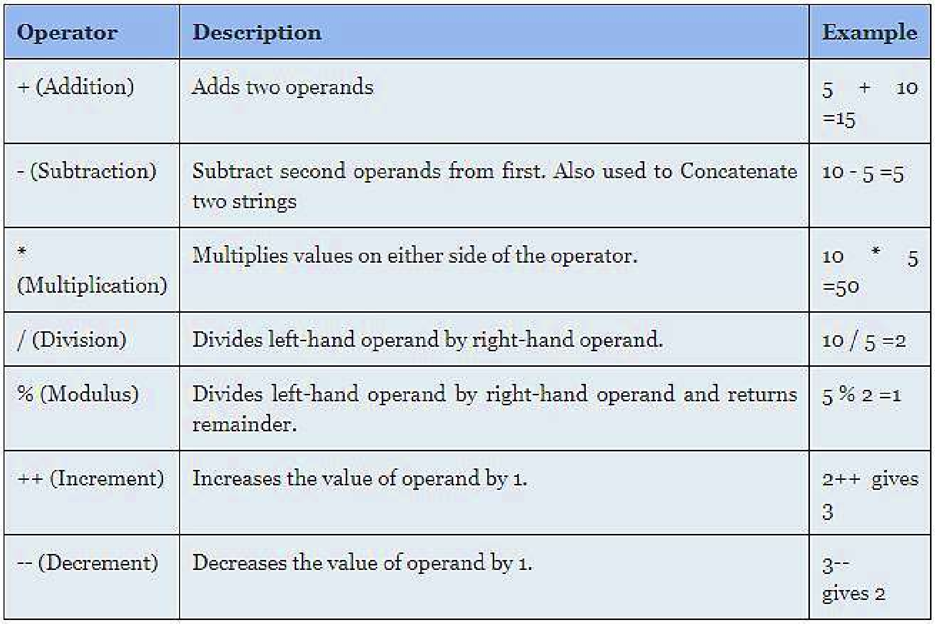


```
32
Press any key to continue . . .
```

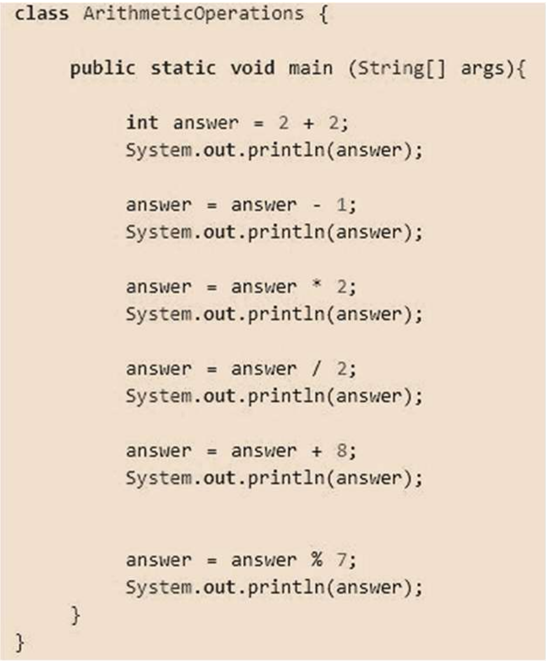
OPERATORS

- An operator is a symbol that tells the computer to perform certain mathematical or logical manipulation.
- Java operators can be divided into following categories:
  - Arithmetic Operators
  - Relational Operators
  - Bitwise Operators
  - Logical Operators
  - Assignment Operators
  - conditional operator (Ternary)

❖Arithmetic Operators



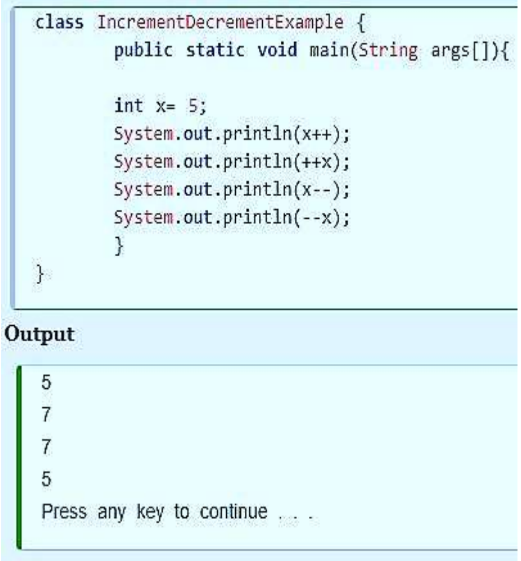
Operator	Description	Example
+ (Addition)	Adds two operands	5 + 10 =15
- (Subtraction)	Subtract second operands from first. Also used to Concatenate two strings	10 - 5 =5
* (Multiplication)	Multiplies values on either side of the operator.	10 * 5 =50
/ (Division)	Divides left-hand operand by right-hand operand.	10 / 5 =2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	5 % 2 =1
++ (Increment)	Increases the value of operand by 1.	2++ gives 3
-- (Decrement)	Decreases the value of operand by 1.	3-- gives 2



```
class ArithmeticOperations {

    public static void main (String[] args){

        int answer = 2 + 2;
        System.out.println(answer);

        answer = answer - 1;
        System.out.println(answer);

        answer = answer * 2;
        System.out.println(answer);

        answer = answer / 2;
        System.out.println(answer);

        answer = answer + 8;
        System.out.println(answer);


        answer = answer % 7;
        System.out.println(answer);
    }
}
```

Output

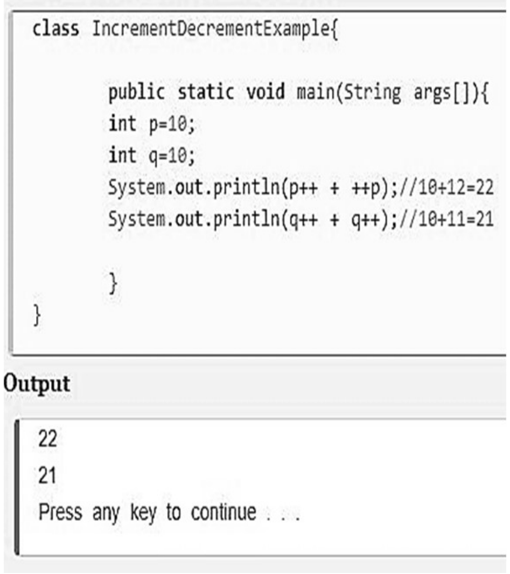
```
4
3
6
3
11
4
```



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

        int x= 5;
        System.out.println(x++);
        System.out.println(++x);
        System.out.println(x--);
        System.out.println(--x);
        }
}
```

Output

```
5
7
7
5
Press any key to continue . . .
```



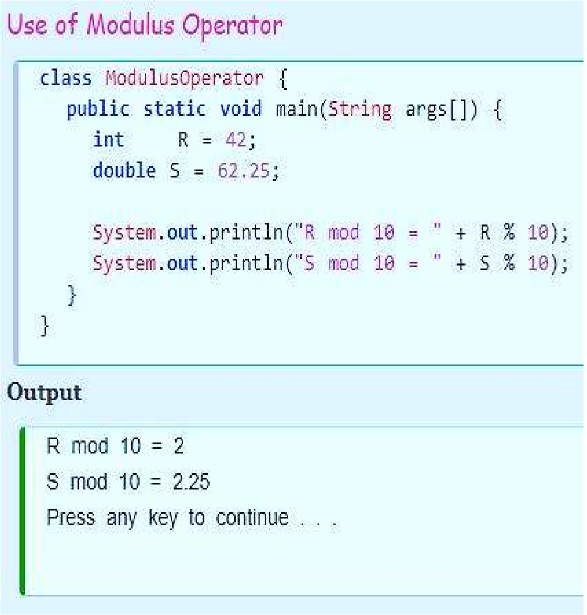
```
class IncrementDecrementExample{

        public static void main(String args[]){
        int p=10;
        int q=10;
        System.out.println(p++ + ++p);//10+12=22
        System.out.println(q++ + q++);//10+11=21

        }
}
```

Output

```
22
21
Press any key to continue . . .
```

X++ is Use –Then - Change

++x is Change – Then - Use

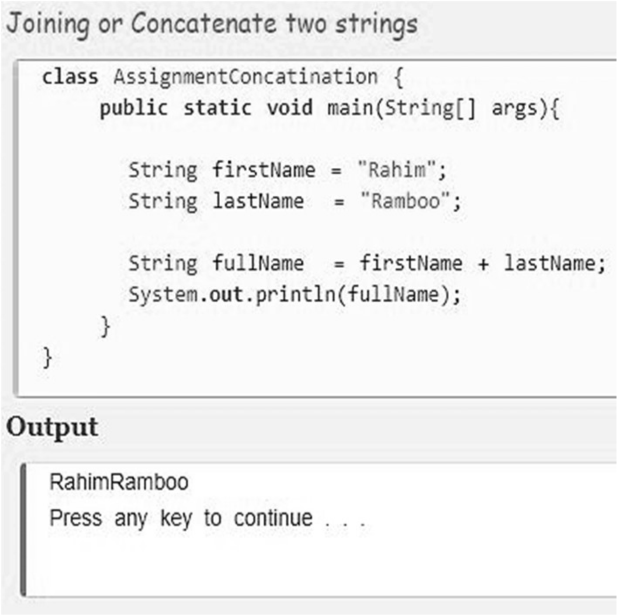


Use of Modulus Operator

```
class ModulusOperator {
   public static void main(String args[]) {
      int     R = 42;
      double S = 62.25;

      System.out.println("R mod 10 = " + R % 10);
      System.out.println("S mod 10 = " + S % 10);
   }
}
```

Output

```
R mod 10 = 2
S mod 10 = 2.25
Press any key to continue . . .
```



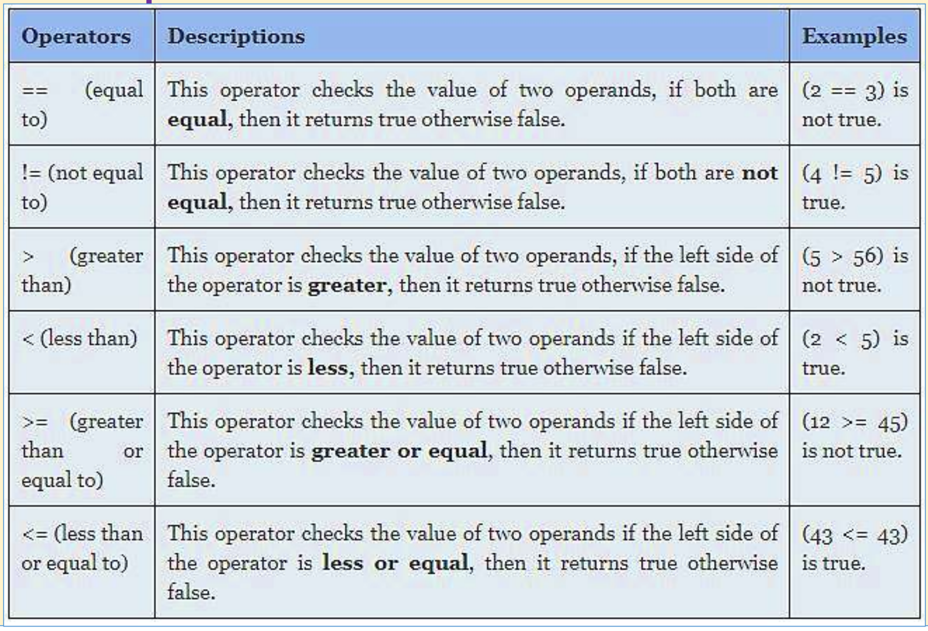
Joining or Concatenate two strings

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

         String firstName  = "Rahim";
         String lastName   = "Ramboo";

         String fullName   = firstName + lastName;
         System.out.println(fullName);
      }
}
```

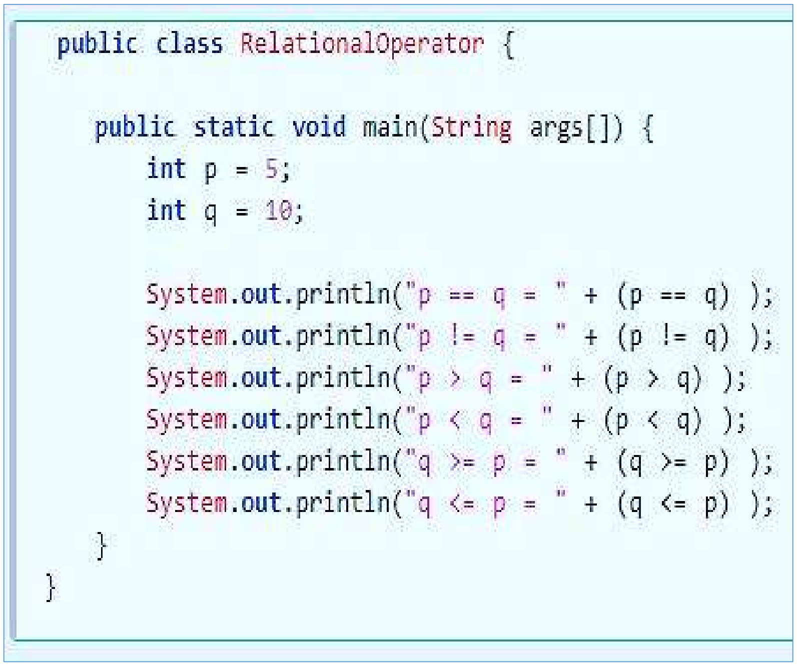
Output

```
RahimRamboo
Press any key to continue . . .
```

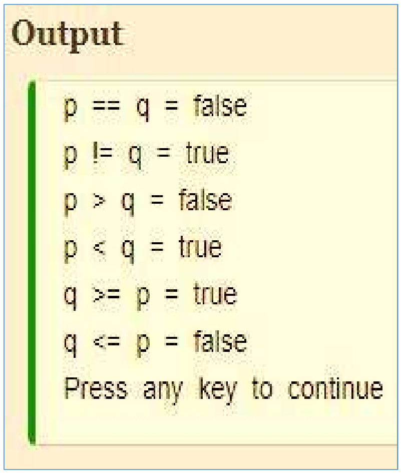
❖Relational Operators



Operators	Descriptions	Examples
== (equal to)	This operator checks the value of two operands, if both are **equal**, then it returns true otherwise false.	(2 == 3) is not true.
!= (not equal to)	This operator checks the value of two operands, if both are **not equal**, then it returns true otherwise false.	(4 != 5) is true.
> (greater than)	This operator checks the value of two operands, if the left side of the operator is **greater**, then it returns true otherwise false.	(5 > 56) is not true.
< (less than)	This operator checks the value of two operands if the left side of the operator is **less**, then it returns true otherwise false.	(2 < 5) is true.
>= (greater than or equal to)	This operator checks the value of two operands if the left side of the operator is **greater or equal**, then it returns true otherwise false.	(12 >= 45) is not true.
<= (less than or equal to)	This operator checks the value of two operands if the left side of the operator is **less or equal**, then it returns true otherwise false.	(43 <= 43) is true.



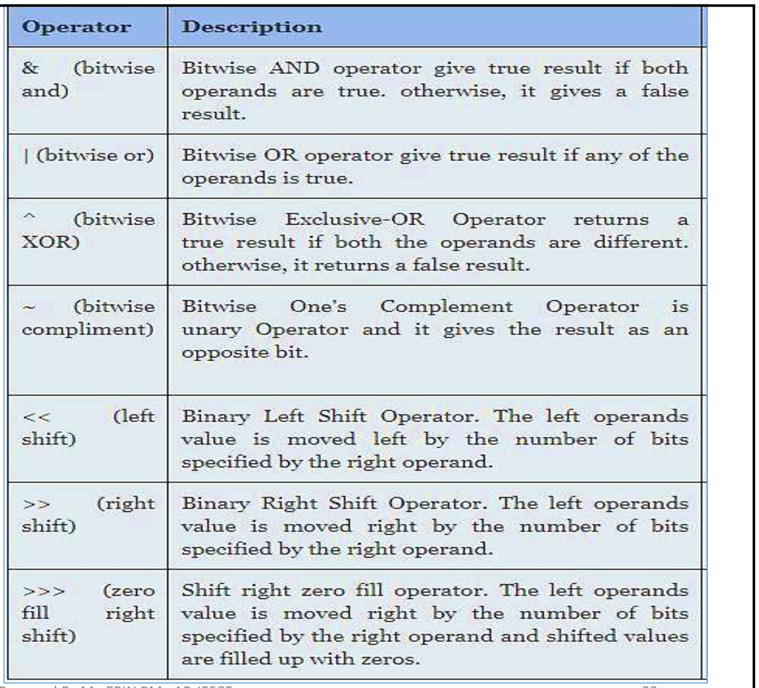
```
public class RelationalOperator {

    public static void main(String args[]) {
        int p = 5;
        int q = 10;

        System.out.println("p == q = " + (p == q) );
        System.out.println("p != q = " + (p != q) );
        System.out.println("p > q = " + (p > q) );
        System.out.println("p < q = " + (p < q) );
        System.out.println("q >= p = " + (q >= p) );
        System.out.println("q <= p = " + (q <= p) );
    }
}
```



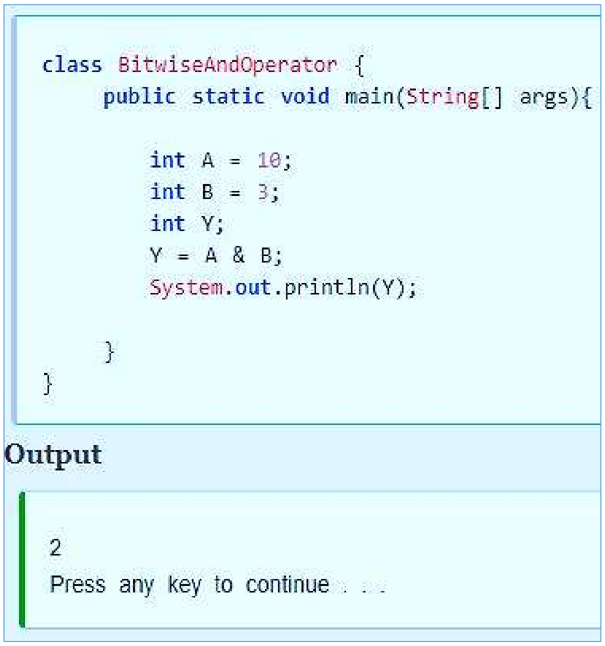
Output

```
p == q = false
p != q = true
p > q = false
p < q = true
q >= p = true
q <= p = false
Press any key to continue
```

❖ Bitwise Operators



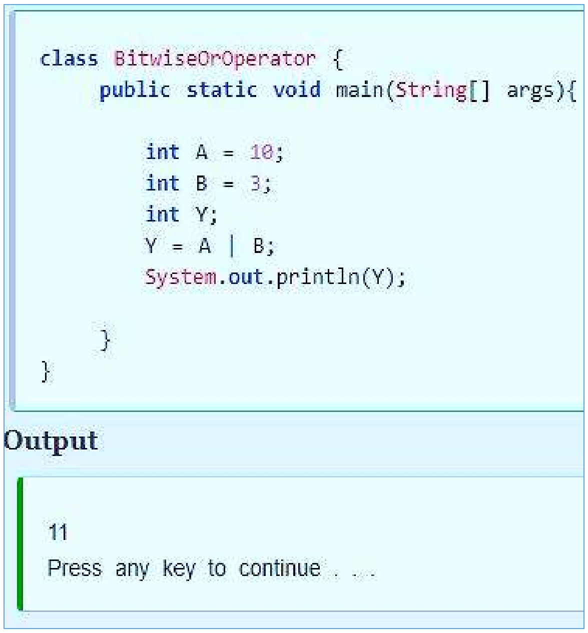
Operator	Description
& (bitwise and)	Bitwise AND operator give true result if both operands are true. otherwise, it gives a false result.
| (bitwise or)	Bitwise OR operator give true result if any of the operands is true.
^ (bitwise XOR)	Bitwise Exclusive-OR Operator returns a true result if both the operands are different. otherwise, it returns a false result.
~ (bitwise compliment)	Bitwise One's Complement Operator is unary Operator and it gives the result as an opposite bit.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.



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

         int A = 10;
         int B = 3;
         int Y;
         Y = A & B;
         System.out.println(Y);

     }
}
```

Output

```
2
Press any key to continue . . .
```

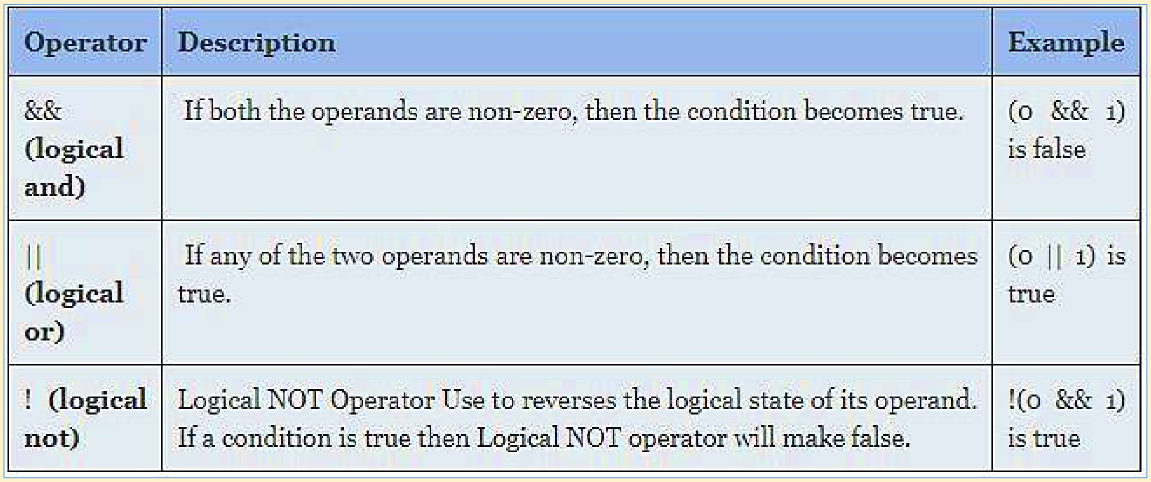


```
class BitwiseOrOperator {
     public static void main(String[] args){

         int A = 10;
         int B = 3;
         int Y;
         Y = A | B;
         System.out.println(Y);

     }
}
```

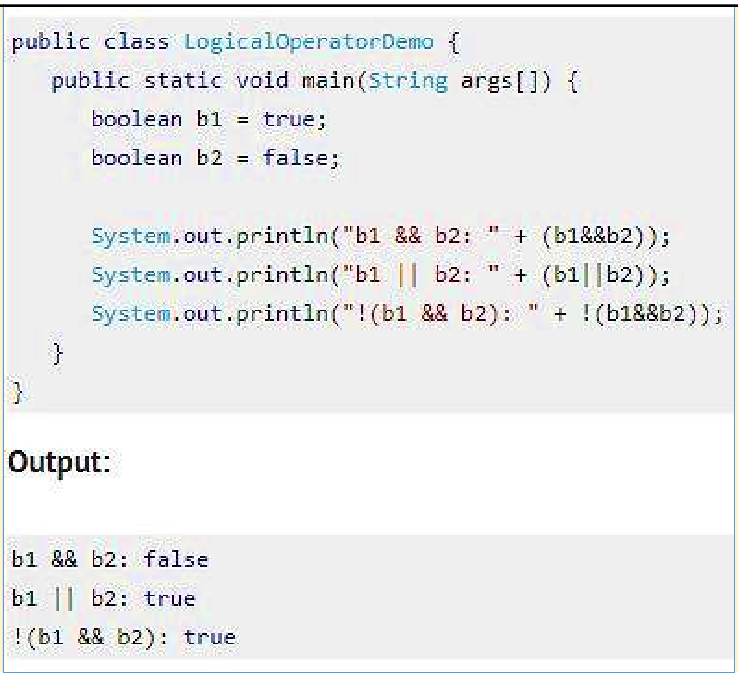
Output

```
11
Press any key to continue . . .
```

❖Logical Operators



Operator	Description	Example
&& **(logical and)**	If both the operands are non-zero, then the condition becomes true.	(0 && 1) is false
|| **(logical or)**	If any of the two operands are non-zero, then the condition becomes true.	(0 || 1) is true
! **(logical not)**	Logical NOT Operator Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(0 && 1) is true

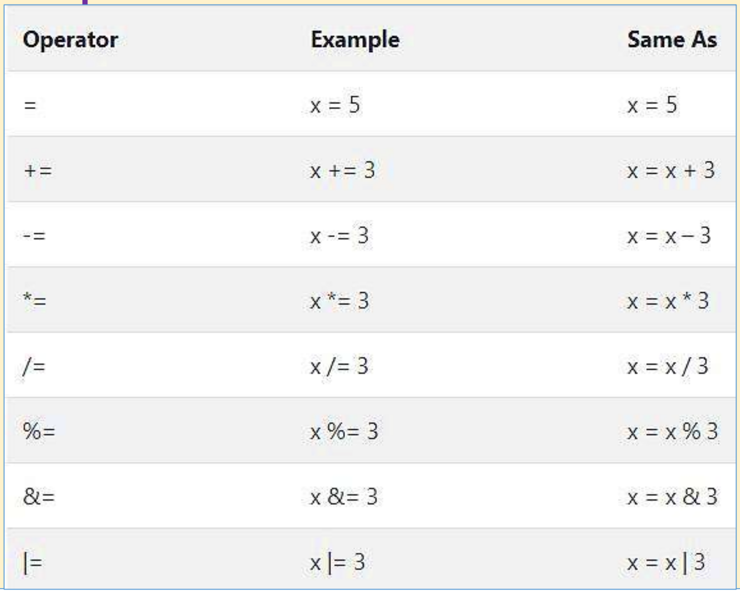


```
public class LogicalOperatorDemo {
   public static void main(String args[]) {
      boolean b1 = true;
      boolean b2 = false;

      System.out.println("b1 && b2: " + (b1&&b2));
      System.out.println("b1 || b2: " + (b1||b2));
      System.out.println("!(b1 && b2): " + !(b1&&b2));
   }
}
```

Output:

```
b1 && b2: false
b1 || b2: true
!(b1 && b2): true
```

❖Assignment Operators

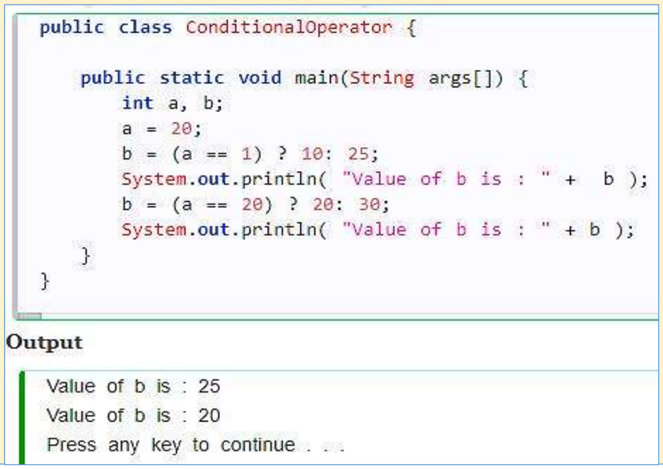


Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3

❖conditional Operator / Ternary Operator (? :)

Expression1 ? Expression2 : Expression3

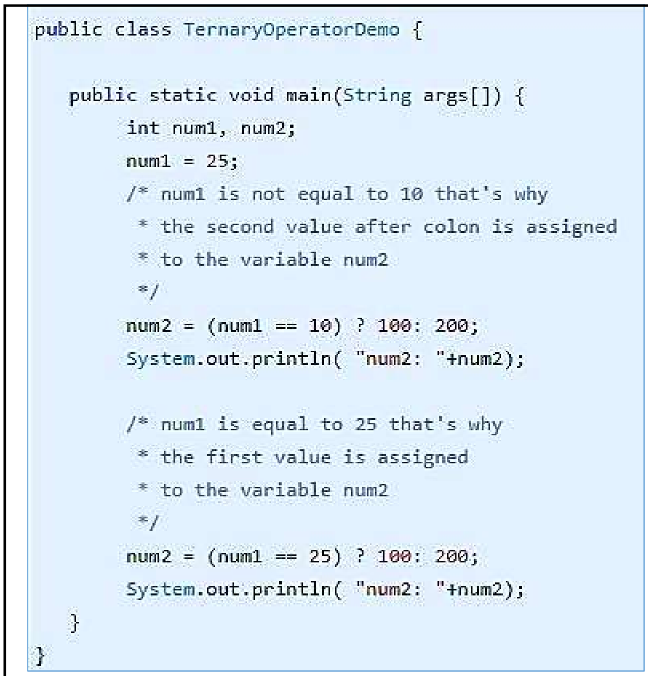
Expression ? value if true : value if false



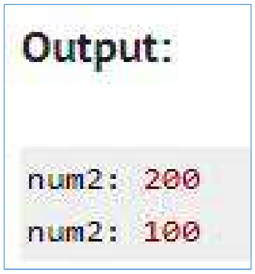
```
public class ConditionalOperator {

    public static void main(String args[]) {
        int a, b;
        a = 20;
        b = (a == 1) ? 10: 25;
        System.out.println( "Value of b is : " +  b );
        b = (a == 20) ? 20: 30;
        System.out.println( "Value of b is : " + b );
    }
}
```

Output

```
Value of b is : 25
Value of b is : 20
Press any key to continue . . .
```



```
public class TernaryOperatorDemo {

   public static void main(String args[]) {
        int num1, num2;
        num1 = 25;
        /* num1 is not equal to 10 that's why
         * the second value after colon is assigned
         * to the variable num2
         */
        num2 = (num1 == 10) ? 100: 200;
        System.out.println( "num2: "+num2);

        /* num1 is equal to 25 that's why
         * the first value is assigned
         * to the variable num2
         */
        num2 = (num1 == 25) ? 100: 200;
        System.out.println( "num2: "+num2);
   }
}
```



Output:

```
num2: 200
num2: 100
```

Operator Precedence

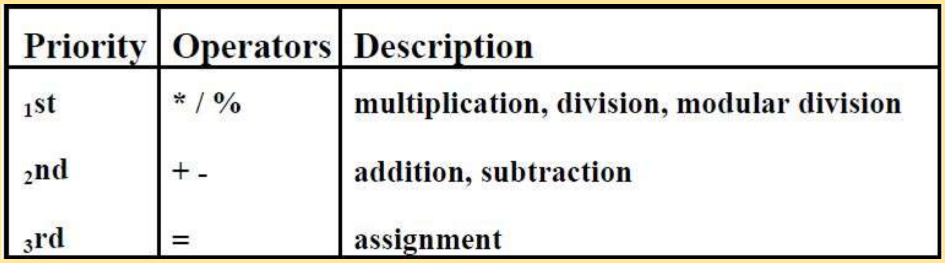
- Evaluate 2*x-3*y ?

  (2x)-(3y) or 2(x-3y) which one is correct??????

- Evaluate A / B * C

  A / (B * C) or (A / B) * C Which one is correct?????

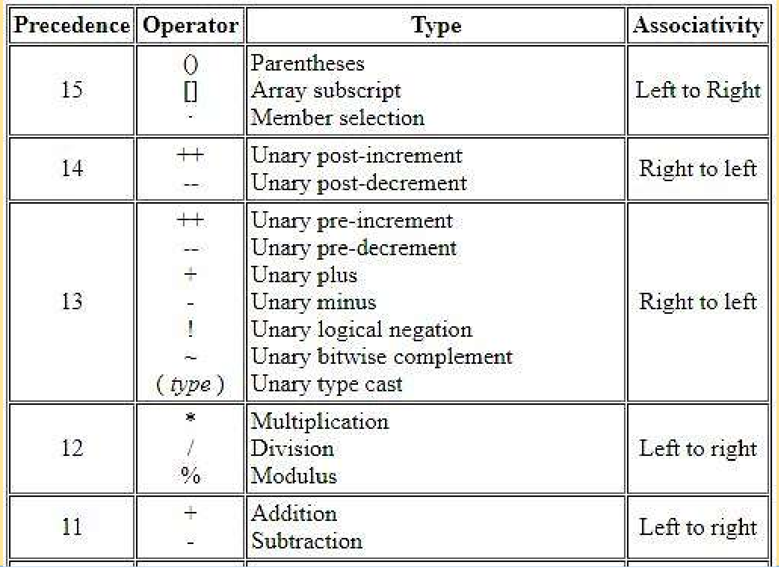
➢ To answer these questions satisfactorily one has to understand the priority or precedence of operations.



Priority	Operators	Description
1st	* / %	multiplication, division, modular division
2nd	+ -	addition, subtraction
3rd	=	assignment

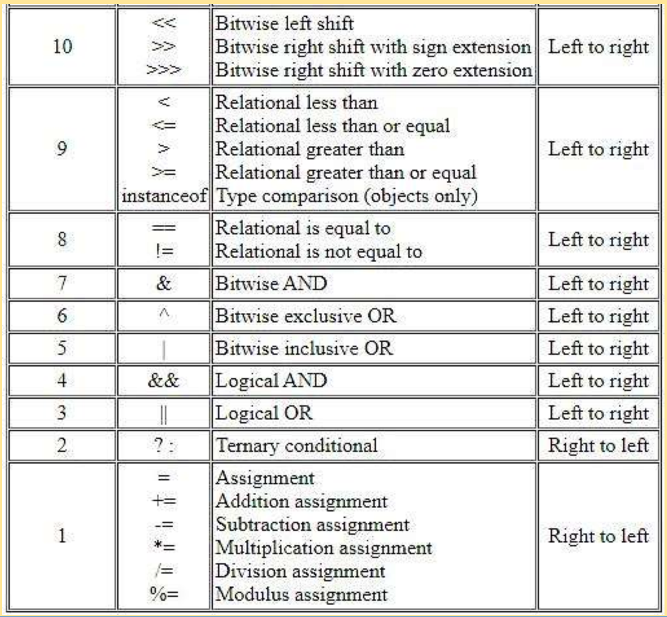
- **Precedence order -** When two operators share an operand the operator with the higher precedence goes first.
- **Associativity -** When an expression has two operators with the same precedence, the expression is evaluated according to its associativity.

➢Larger number means higher precedence



Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (*type*)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right

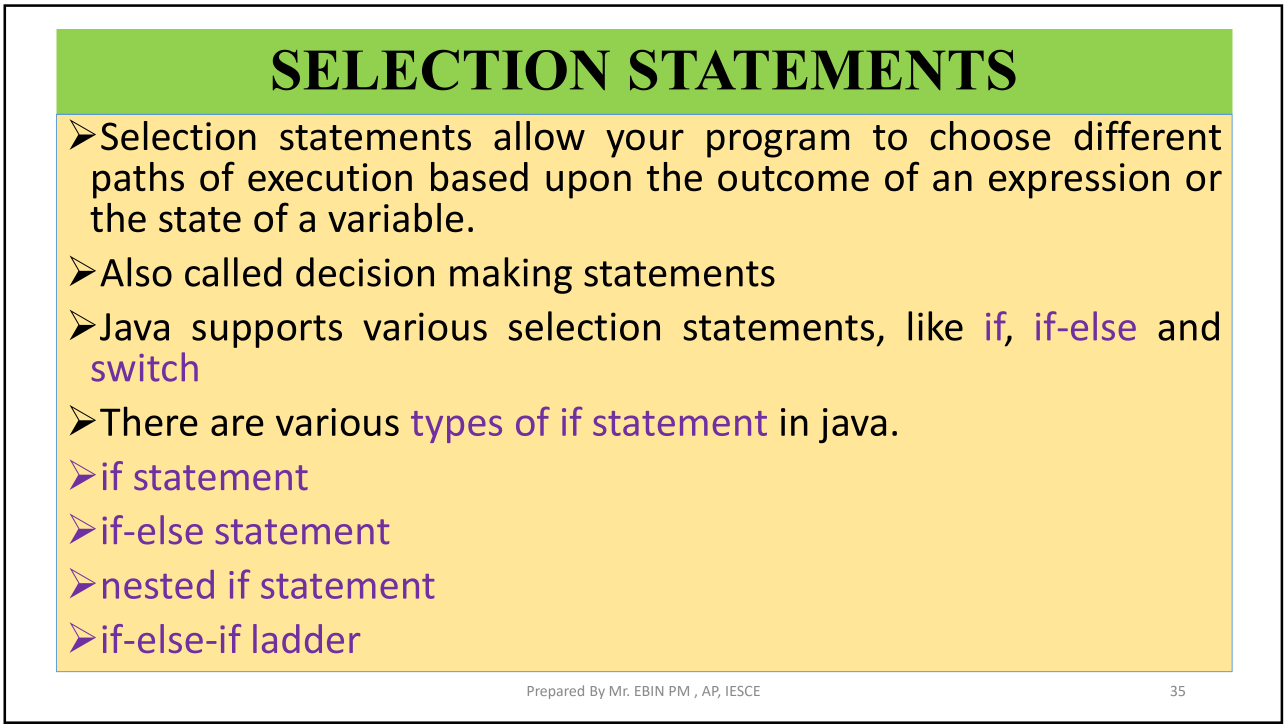
➢Larger number means higher precedence



10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

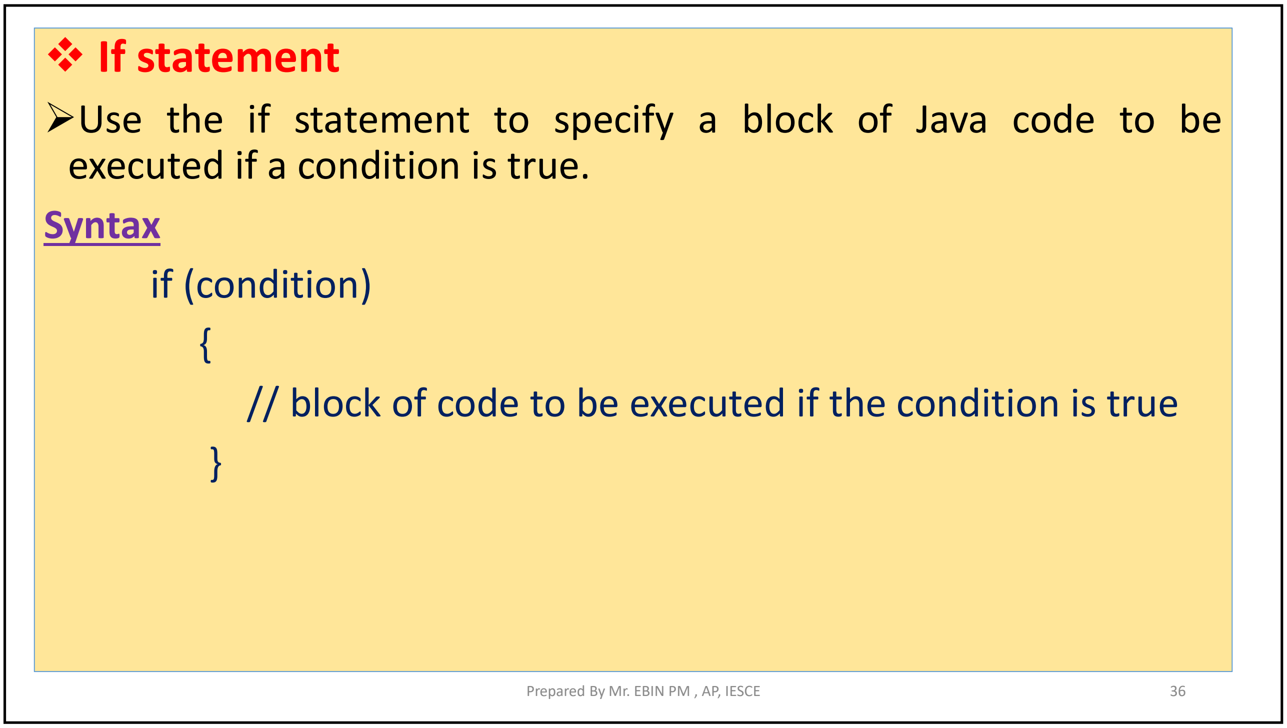
➢Evaluate **i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8**

i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8	**operation: ***
i = 1 + 4 / 4 + 8 - 2 + 5 / 8	**operation: /**
i = 1 + 1+ 8 - 2 + 5 / 8	**operation: /**
i = 1 + 1+ 8 - 2 + 0	**operation: /**
i = 2 + 8- 2 + 0	**operation: +**
i = 10 - 2 + 0	**operation: +**
i = 8 + 0	**operation : -**
i = 8	**operation: +**



SELECTION STATEMENTS

- ➢Selection statements allow your program to choose different paths of execution based upon the outcome of an expression or the state of a variable.
- ➢Also called decision making statements
- ➢Java supports various selection statements, like if, if-else and switch
- ➢There are various types of if statement in java.
- ➢if statement
- ➢if-else statement
- ➢nested if statement
- ➢if-else-if ladder



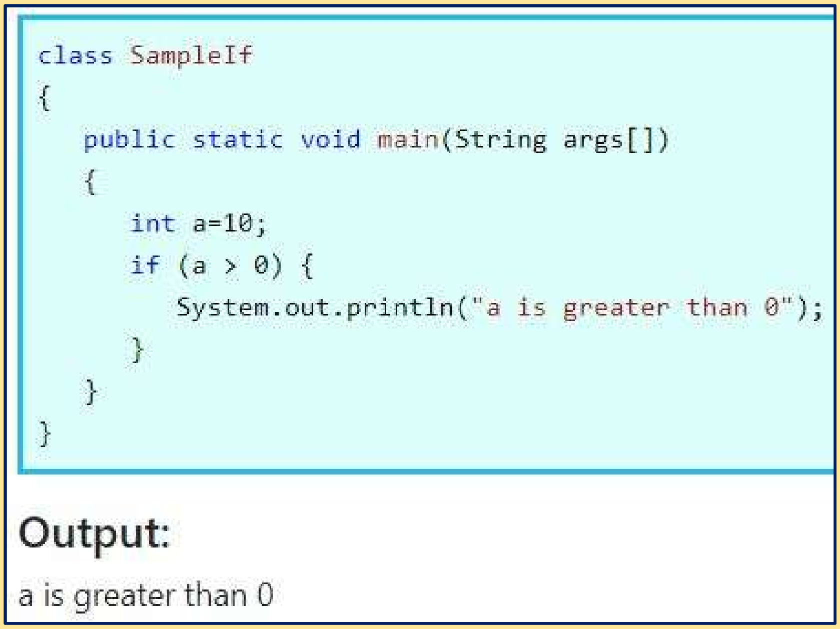
❖ If statement

- ➢Use the if statement to specify a block of Java code to be executed if a condition is true.

Syntax

```
        if (condition)
            {
                // block of code to be executed if the condition is true
             }
```

Example



```
class SampleIf
{
    public static void main(String args[])
    {
        int a=10;
        if (a > 0) {
            System.out.println("a is greater than 0");
        }
    }
}
```

Output:

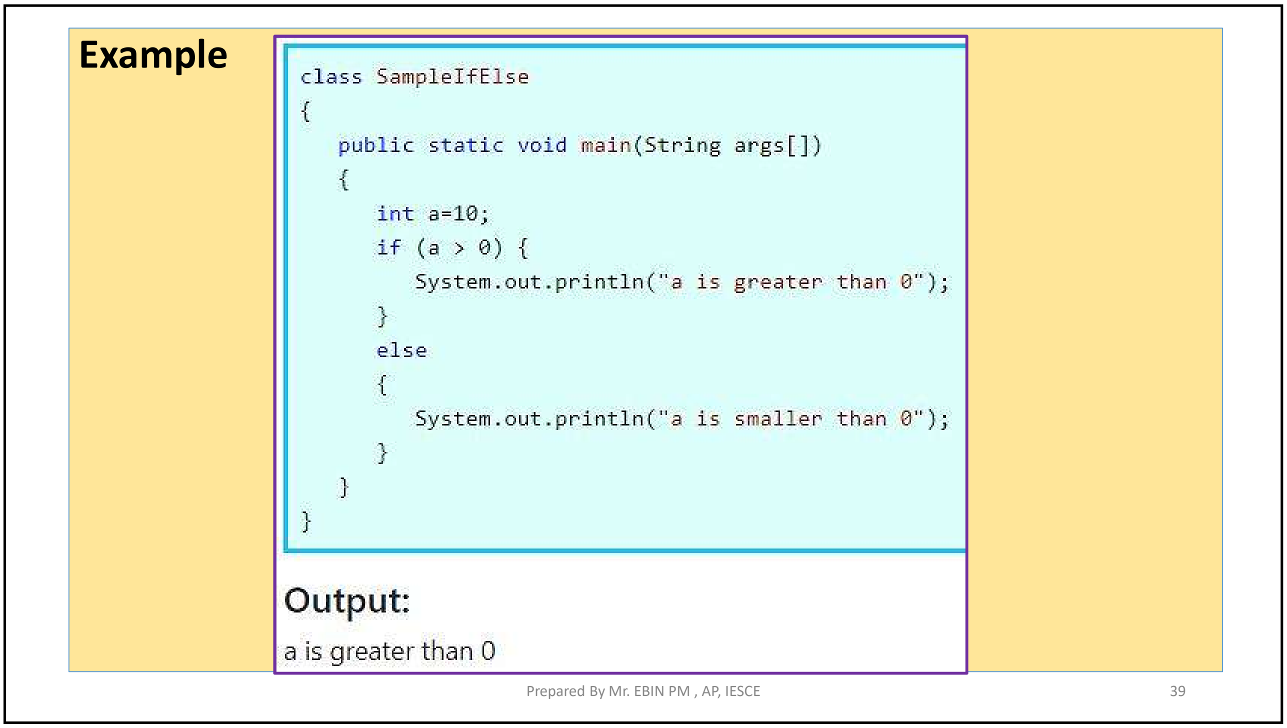
```
a is greater than 0
```

❖ if-else Statement

➢If-else statement also tests the condition. It executes the if block if condition is true otherwise else block is executed.

Syntax

```
 if (condition)
    {
        // block of code to be executed if the condition is true
    }
else
   {
        // block of code to be executed if the condition is false
   }
```

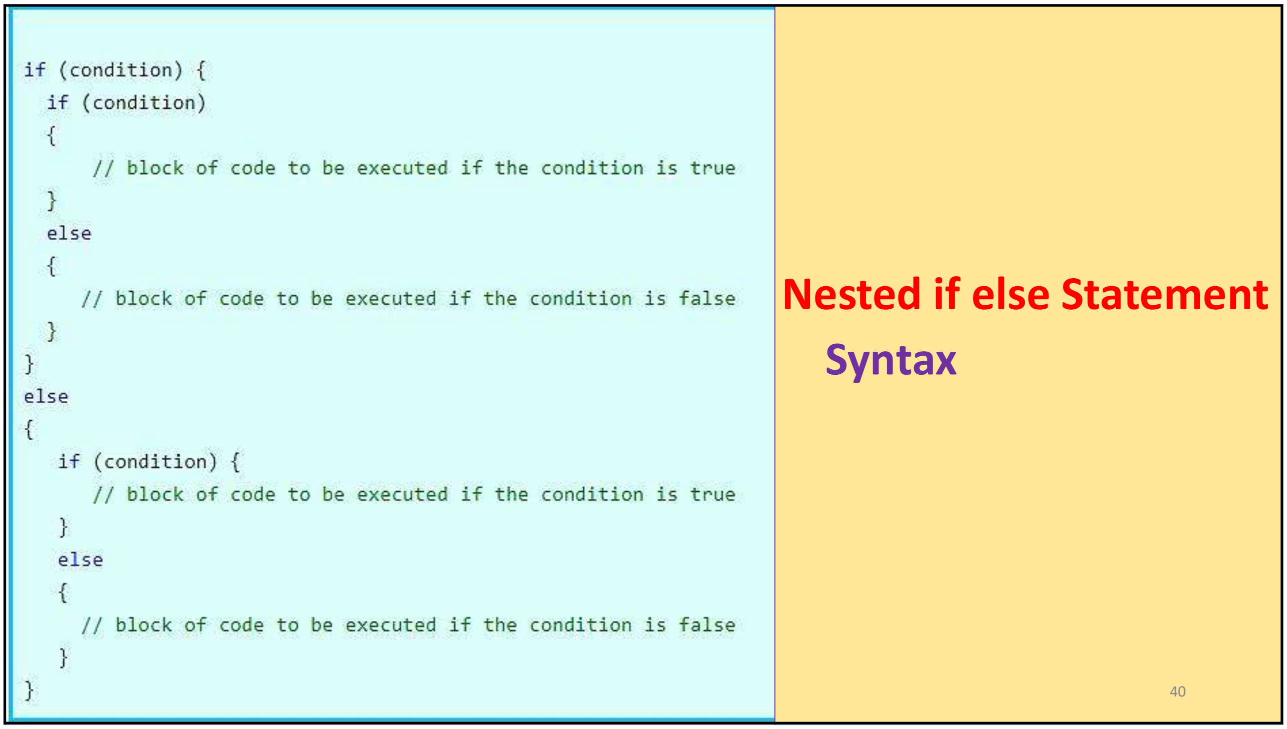


Example

```
class SampleIfElse
{
   public static void main(String args[])
   {
      int a=10;
      if (a > 0) {
         System.out.println("a is greater than 0");
      }
      else
      {
         System.out.println("a is smaller than 0");
      }
   }
}
```

Output:

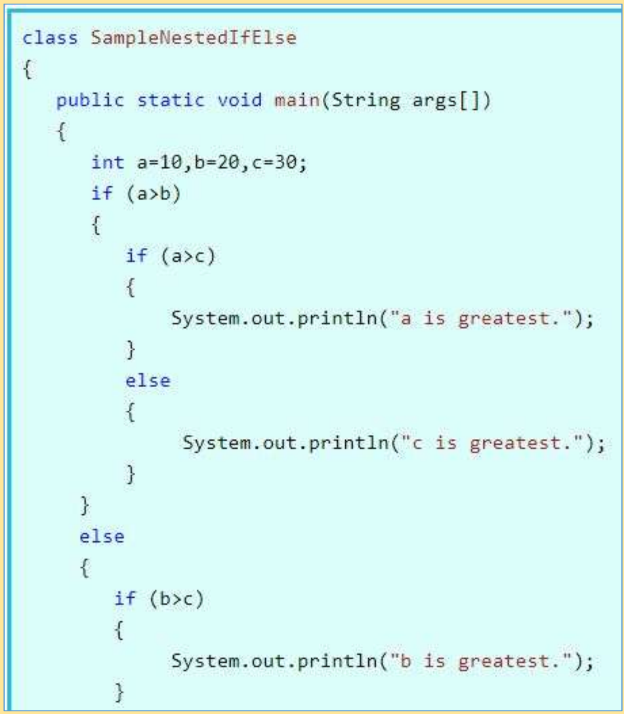
```
a is greater than 0
```



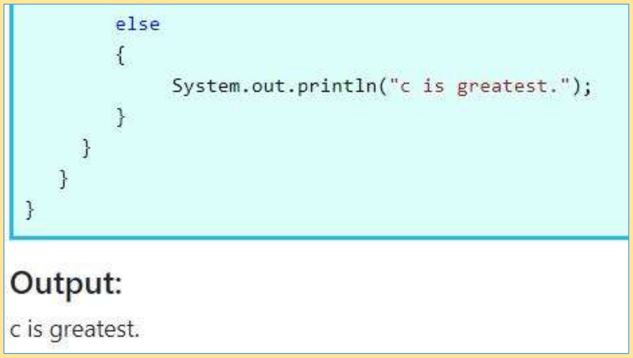
Nested if else Statement

Syntax

```
if (condition) {
  if (condition)
  {
     // block of code to be executed if the condition is true
  }
  else
  {
    // block of code to be executed if the condition is false
  }
}
else
{
   if (condition) {
     // block of code to be executed if the condition is true
   }
   else
   {
    // block of code to be executed if the condition is false
   }
}
```



```
class SampleNestedIfElse
{
   public static void main(String args[])
   {
      int a=10,b=20,c=30;
      if (a>b)
      {
         if (a>c)
         {
             System.out.println("a is greatest.");
         }
         else
         {
              System.out.println("c is greatest.");
         }
     }
     else
     {
        if (b>c)
        {
             System.out.println("b is greatest.");
        }
```



```
         else
         {
              System.out.println("c is greatest.");
         }
      }
    }
 }
```

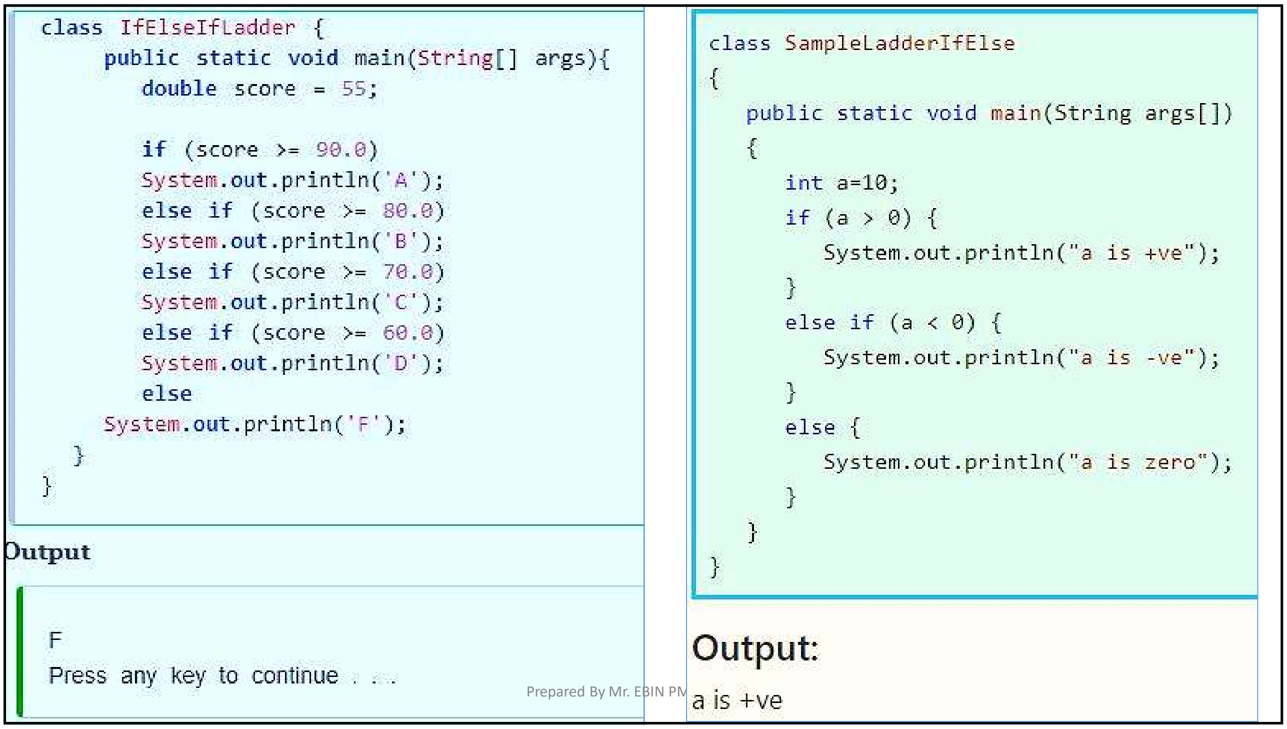
Output:

```
c is greatest.
```

❖ if else if ladder

Syntax

```
if (condition)
 {
   // block of code to be executed if the condition is true
 }
else if (condition)
 {
   // block of code to be executed if the condition is true
 }
else
 {
   // block of code to be executed if the condition is true
 }
```



```
class IfElseIfLadder {
     public static void main(String[] args){
        double score = 55;

        if (score >= 90.0)
        System.out.println('A');
        else if (score >= 80.0)
        System.out.println('B');
        else if (score >= 70.0)
        System.out.println('C');
        else if (score >= 60.0)
        System.out.println('D');
        else
     System.out.println('F');
   }
}
```

Output

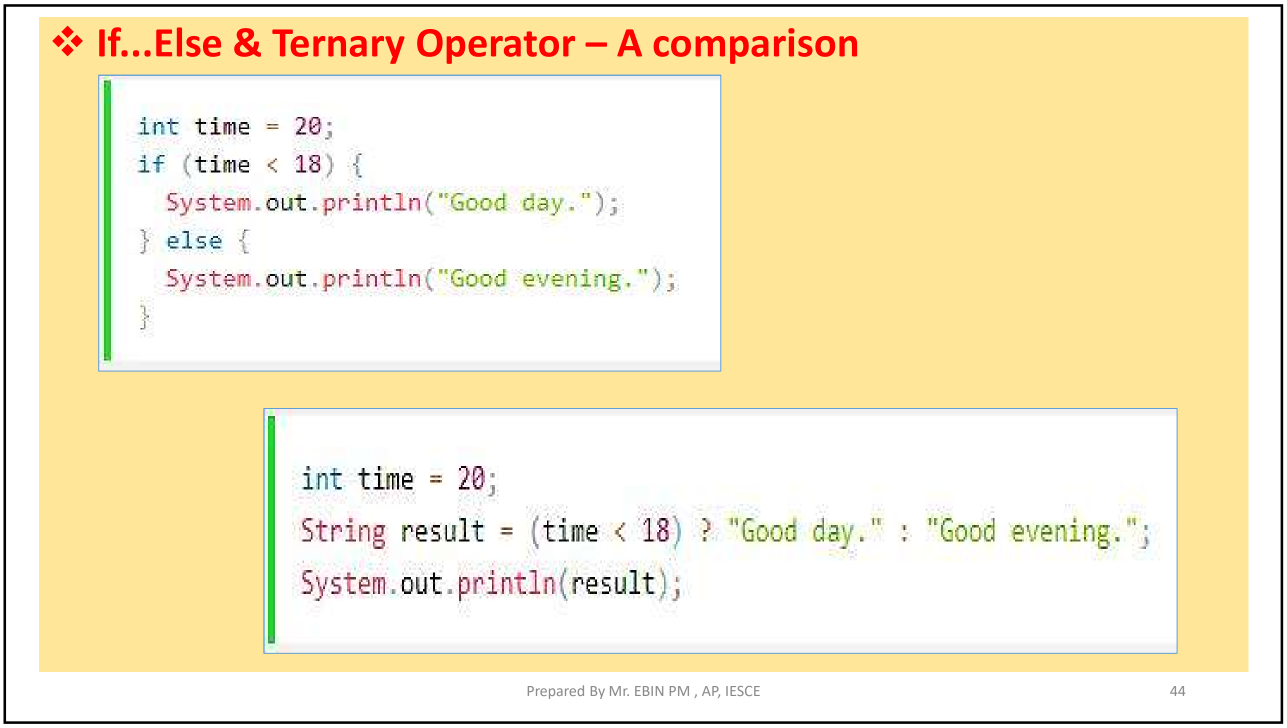
```
F
Press any key to continue . . .
```

```
class SampleLadderIfElse
{
   public static void main(String args[])
   {
      int a=10;
      if (a > 0) {
         System.out.println("a is +ve");
      }
      else if (a < 0) {
         System.out.println("a is -ve");
      }
      else {
         System.out.println("a is zero");
      }
   }
}
```

Output:

```
a is +ve
```

Prepared By Mr. EBIN PM



❖ If...Else & Ternary Operator – A comparison

```
int time = 20;
if (time < 18) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
```

```
int time = 20;
String result = (time < 18) ? "Good day." : "Good evening.";
System.out.println(result);
```

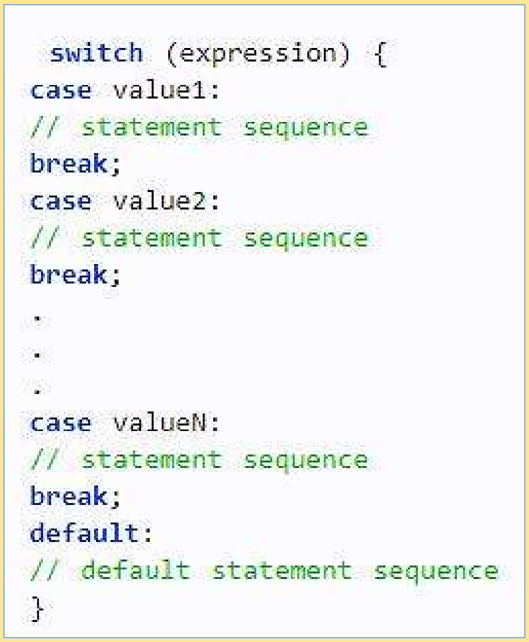
Prepared By Mr. EBIN PM , AP, IESCE 44

❖ switch case

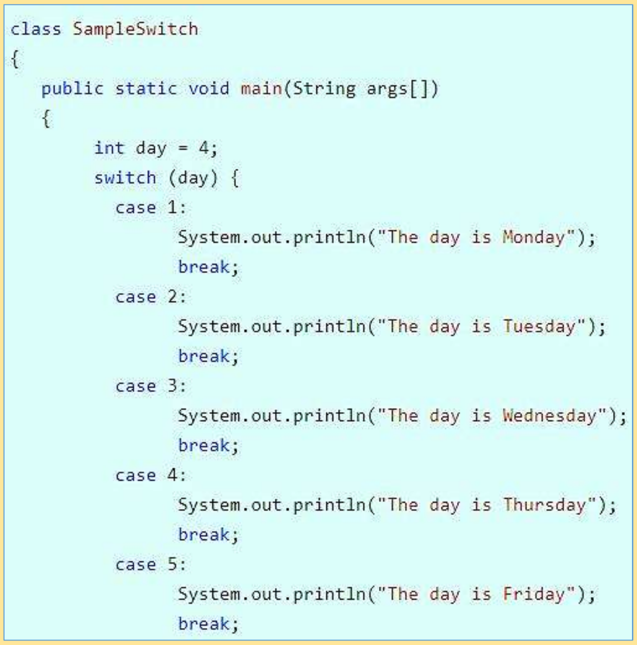
- The if statement in java, makes selections based on a single true or false condition. But switch case have multiple choice for selection of the statements
- It is like if-else-if ladder statement
- **How to Java switch works:**

- Matching each expression with case
- Once it match, execute all case from where it matched.
- Use break to exit from switch
- Use default when expression does not match with any case

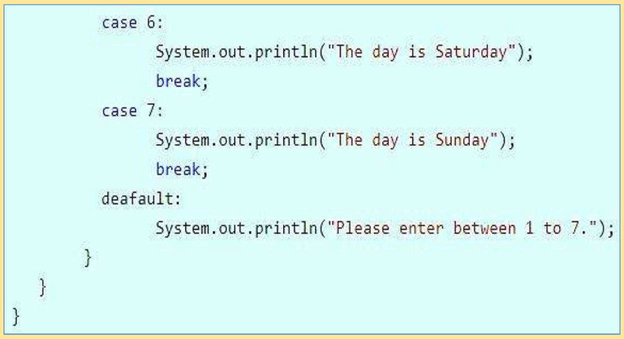
Syntax



```
 switch (expression) {
case value1:
// statement sequence
break;
case value2:
// statement sequence
break;
.
.
.
case valueN:
// statement sequence
break;
default:
// default statement sequence
}
```



```
class SampleSwitch
{
   public static void main(String args[])
   {
        int day = 4;
        switch (day) {
          case 1:
                System.out.println("The day is Monday");
                break;
          case 2:
                System.out.println("The day is Tuesday");
                break;
          case 3:
                System.out.println("The day is Wednesday");
                break;
          case 4:
                System.out.println("The day is Thursday");
                break;
          case 5:
                System.out.println("The day is Friday");
                break;
```



```
          case 6:
                System.out.println("The day is Saturday");
                break;
          case 7:
                System.out.println("The day is Sunday");
                break;
          deafault:
                System.out.println("Please enter between 1 to 7.");
        }
   }
}
```

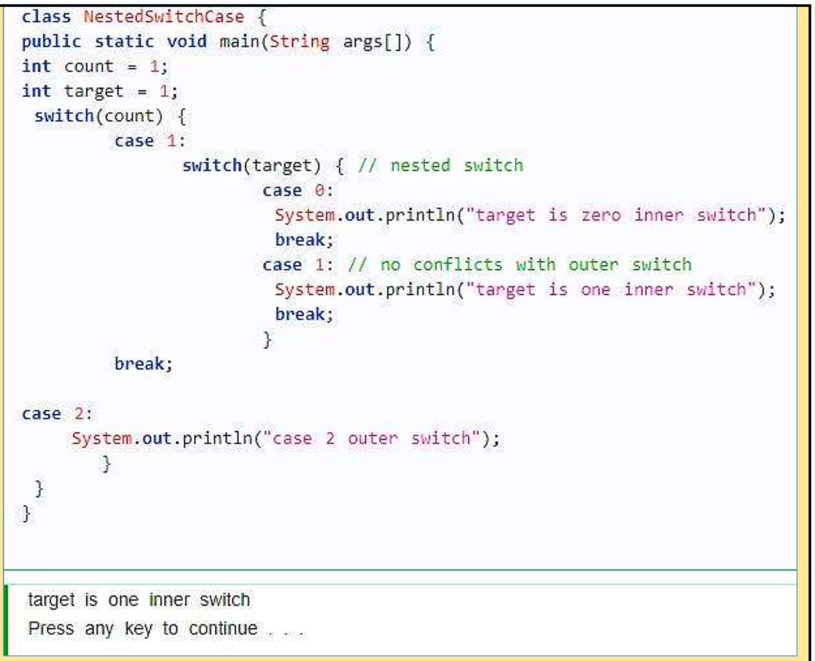
Output

The day is Thursday

Why break is necessary in switch statement ?

- The break statement is used inside the switch to terminate a statement sequence.
- When a break statement is encountered, execution branches to the first line of code that follows the entire switch statement
- This has the effect of jumping out of the switch.
- The break statement is optional. If you omit the break, execution will continue on into the next case.

Nested Switch



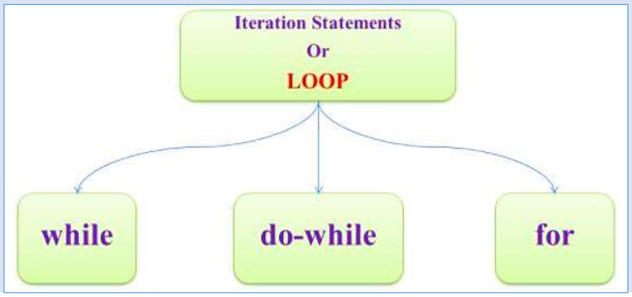
```
class NestedSwitchCase {
public static void main(String args[]) {
int count = 1;
int target = 1;
 switch(count) {
        case 1:
              switch(target) { // nested switch
                     case 0:
                      System.out.println("target is zero inner switch");
                      break;
                     case 1: // no conflicts with outer switch
                      System.out.println("target is one inner switch");
                      break;
                     }
        break;

case 2:
    System.out.println("case 2 outer switch");
      }
 }
}
```

```
target is one inner switch
Press any key to continue . . .
```

Prepared By Mr. EBIN PM , AP, IESCE 49

Iteration Statements (Loop)

- A loop can be used to tell a program to execute statements repeatedly
- A loop repeatedly executes the same set of instructions until a termination condition is met.



Prepared By Mr. EBIN PM , AP, IESCE 50

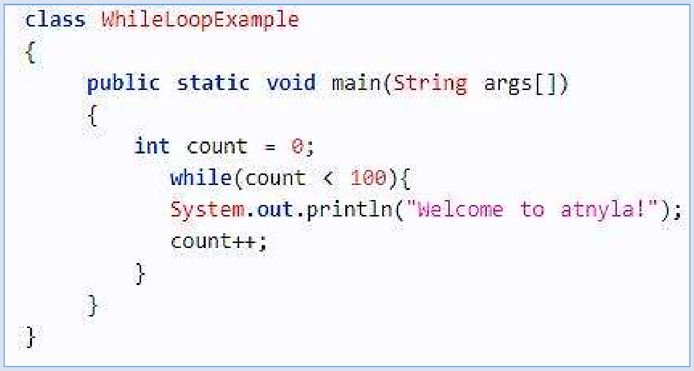
❖ While Loop

➢In **while loop** first checks the condition if the condition is true then control goes inside the loop body otherwise goes outside of the body.

Syntax

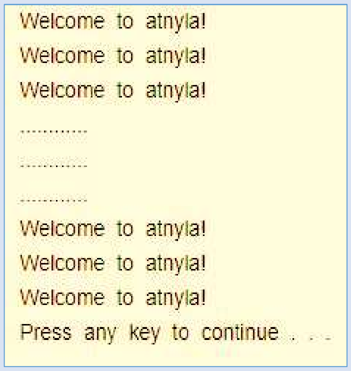
```
while (condition)
 {
   // code block to be executed
 }
```

Example - 1

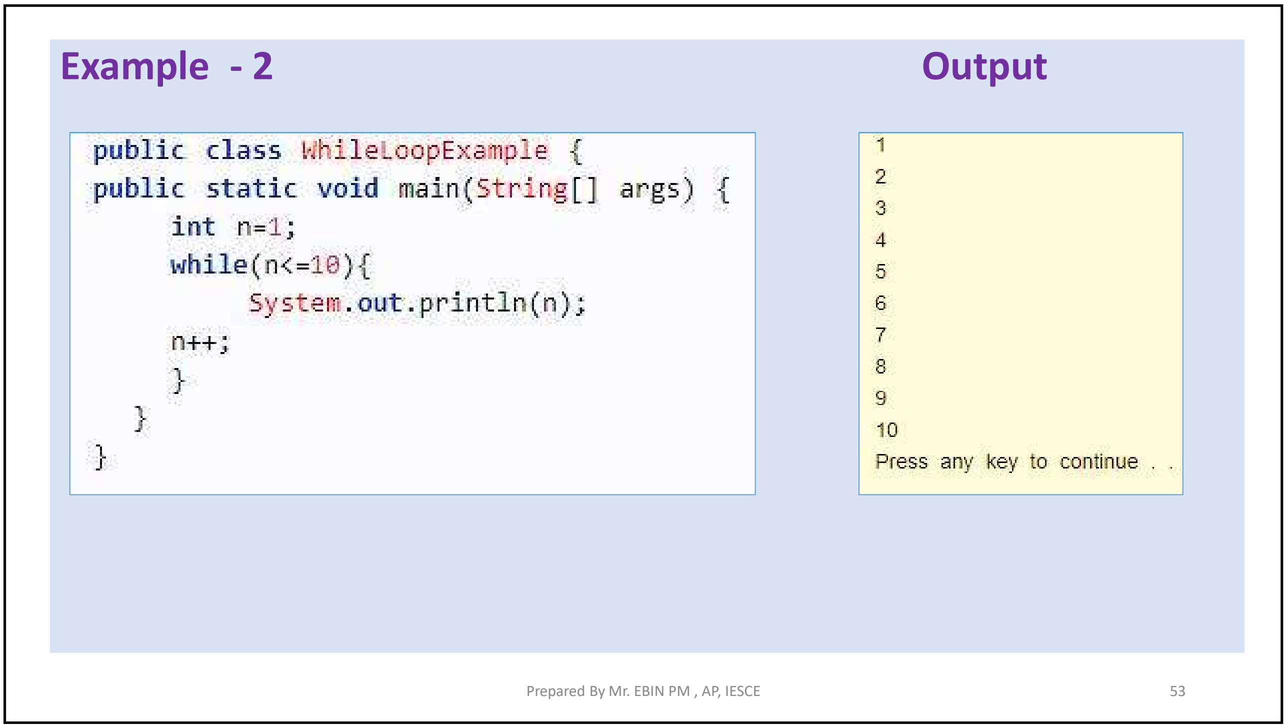


```
class WhileLoopExample
{
     public static void main(String args[])
     {
         int count = 0;
            while(count < 100){
            System.out.println("Welcome to atnyla!");
            count++;
         }
     }
}
```

Output



```
Welcome to atnyla!
Welcome to atnyla!
Welcome to atnyla!
............
............
............
Welcome to atnyla!
Welcome to atnyla!
Welcome to atnyla!
Press any key to continue . . .
```



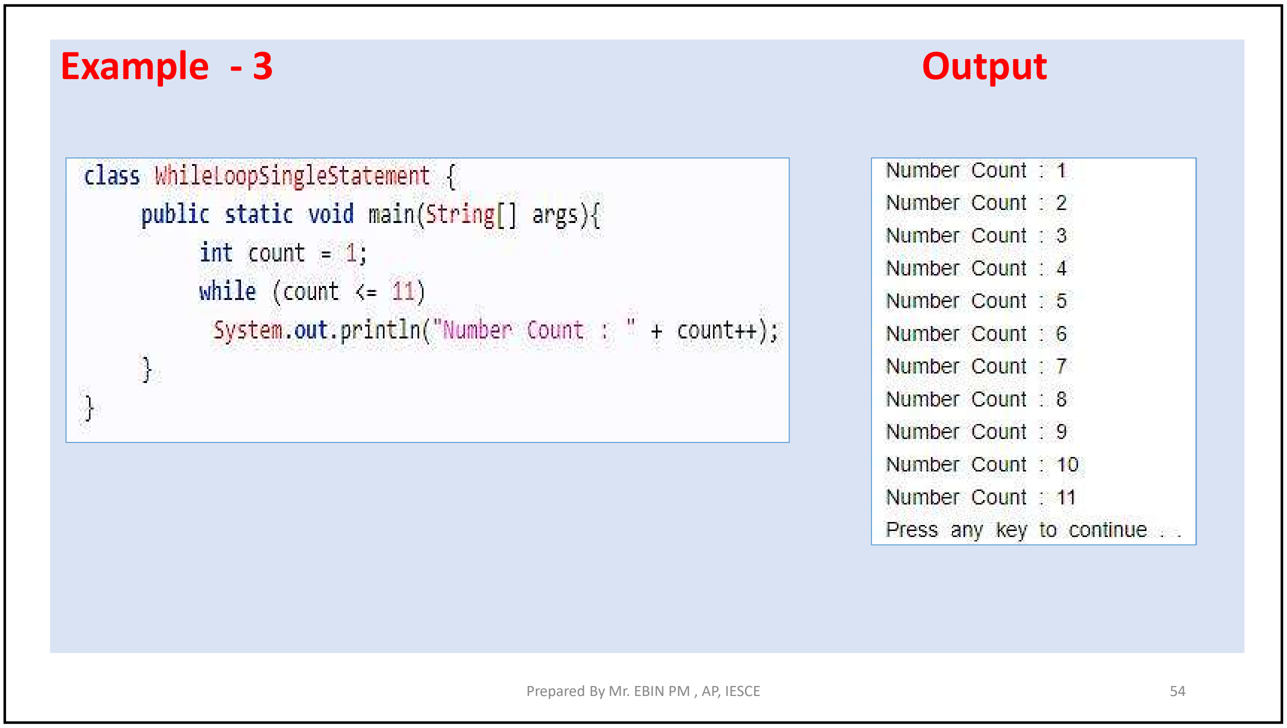
Example - 2

Output

```
public class WhileLoopExample {
public static void main(String[] args) {
      int n=1;
      while(n<=10){
            System.out.println(n);
      n++;
      }
   }
}
```

```
1
2
3
4
5
6
7
8
9
10
Press any key to continue . .
```

Prepared By Mr. EBIN PM , AP, IESCE 53



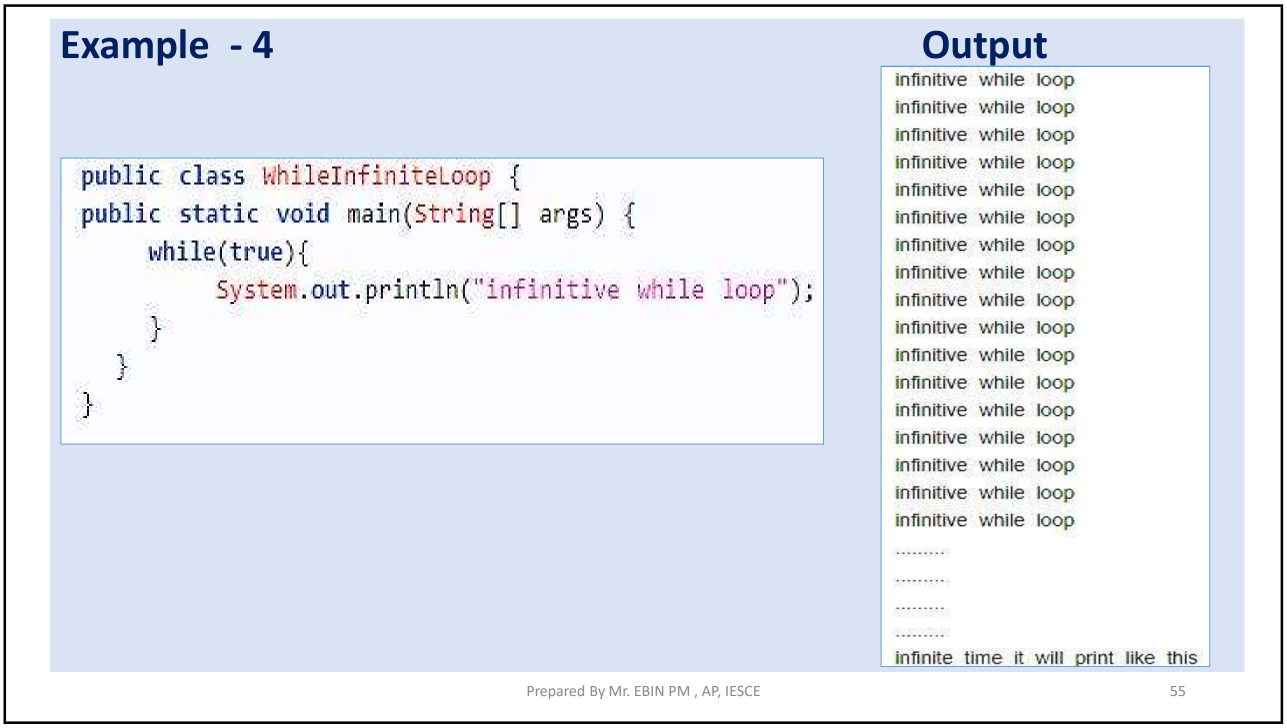
Example - 3

Output

```
class WhileLoopSingleStatement {
      public static void main(String[] args){
            int count = 1;
            while (count <= 11)
             System.out.println("Number Count : " + count++);
      }
}
```

```
Number Count : 1
Number Count : 2
Number Count : 3
Number Count : 4
Number Count : 5
Number Count : 6
Number Count : 7
Number Count : 8
Number Count : 9
Number Count : 10
Number Count : 11
Press any key to continue . .
```

Prepared By Mr. EBIN PM , AP, IESCE 54

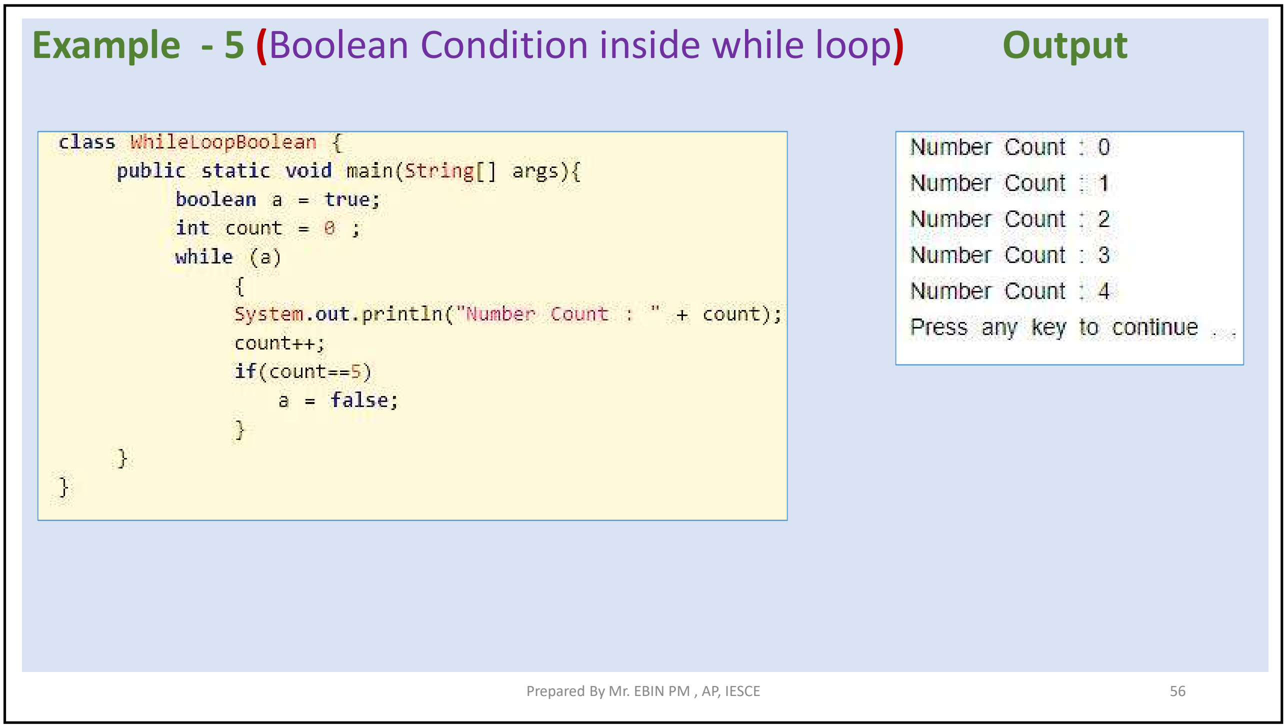


Example - 4

```
public class WhileInfiniteLoop {
public static void main(String[] args) {
      while(true){
            System.out.println("infinitive while loop");
      }
   }
}
```

Output

```
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
.........
.........
.........
.........
infinite time it will print like this
```



Example - 5 (Boolean Condition inside while loop)

```
class WhileLoopBoolean {
     public static void main(String[] args){
          boolean a = true;
          int count = 0 ;
          while (a)
               {
               System.out.println("Number Count : " + count);
               count++;
               if(count==5)
                   a = false;
               }
     }
}
```

Output

```
Number Count : 0
Number Count : 1
Number Count : 2
Number Count : 3
Number Count : 4
Press any key to continue . .
```

❖ do...while loop

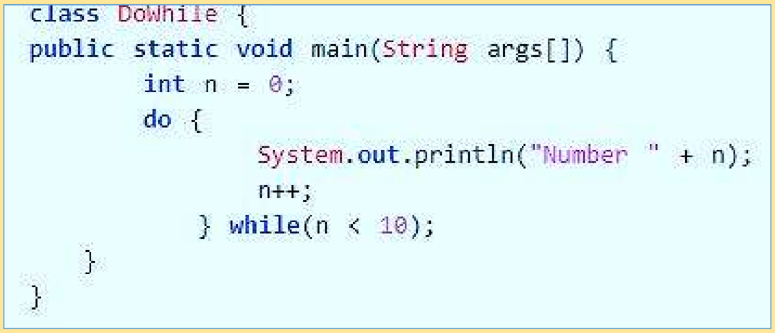
➢ A do while loop is a control flow statement that executes a block of code at least once, and then repeatedly executes the block, or not, depending on a given condition at the end of the block (in while).

Syntax

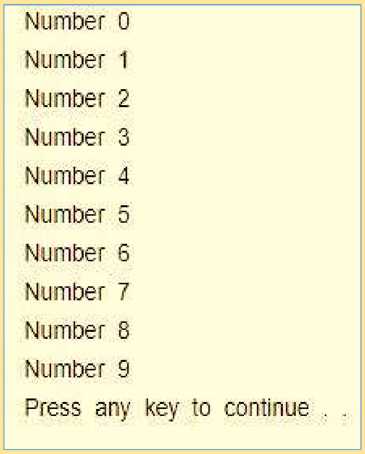
```
do {
        // code block to be executed
    } while (condition);
```

Example -1

Output

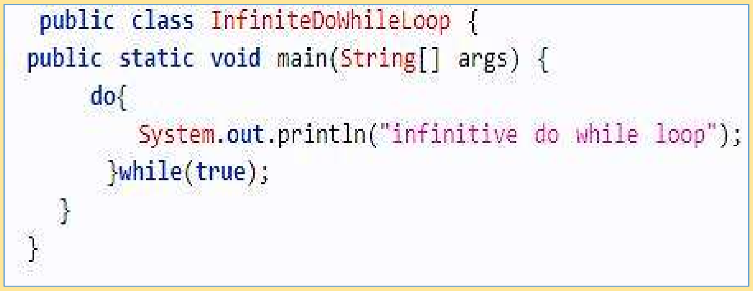


```
class DoWhile {
public static void main(String args[]) {
        int n = 0;
        do {
                System.out.println("Number " + n);
                n++;
            } while(n < 10);
    }
}
```



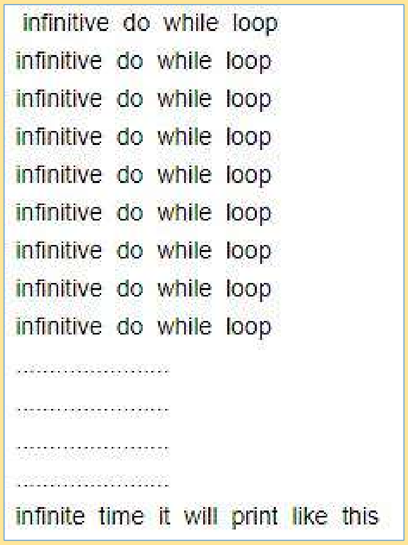
```
Number 0
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
Press any key to continue . . .
```

Example -2 (Infinitive do-while Loop)



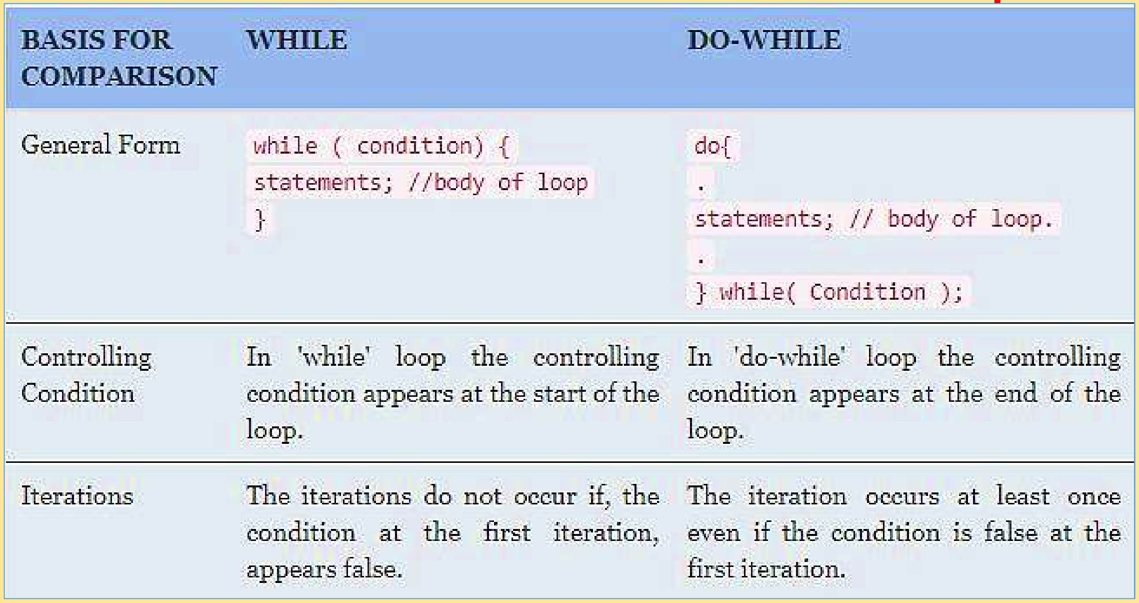
```
 public class InfiniteDoWhileLoop {
public static void main(String[] args) {
     do{
          System.out.println("infinitive do while loop");
      }while(true);
   }
}
```

Output



```
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
........................
........................
........................
........................
infinite time it will print like this
```

Difference Between while and do-while Loop



BASIS FOR COMPARISON	WHILE	DO-WHILE

General Form	`while ( condition) {`
	`statements; //body of loop`
	`}`
		`do{`
		`.`
		`statements; // body of loop.`
		`.`
		`} while( Condition );`

Controlling Condition	In 'while' loop the controlling condition appears at the start of the loop.	In 'do-while' loop the controlling condition appears at the end of the loop.
Iterations	The iterations do not occur if, the condition at the first iteration, appears false.	The iteration occurs at least once even if the condition is false at the first iteration.

❖ for loop

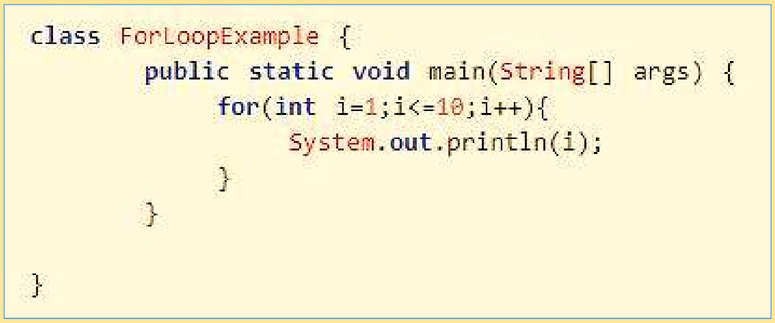
➢For Loop is used to execute set of statements repeatedly until the condition is true.

Syntax

```
for (initialization; condition; increment/decrement)
  {
    // code block to be executed
  }
```

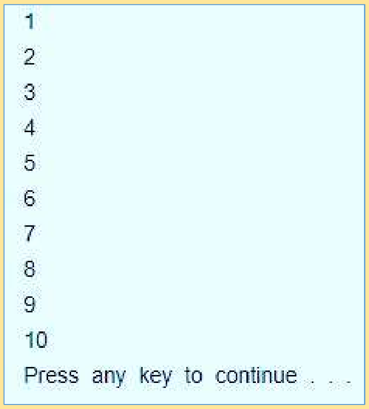
Initialization : It executes at once.
Condition : This check until get true.
Increment/Decrement: This is for increment or decrement.

Example 1



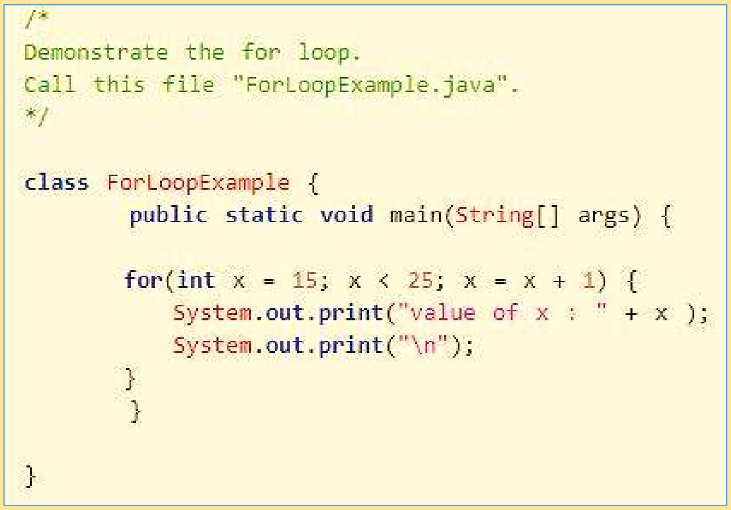
```
class ForLoopExample {
        public static void main(String[] args) {
            for(int i=1;i<=10;i++){
                System.out.println(i);
            }
        }

}
```

Output



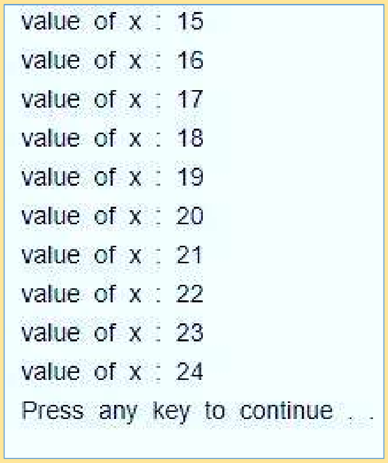
```
1
2
3
4
5
6
7
8
9
10
Press any key to continue . . .
```

Example 2



```
/*
Demonstrate the for loop.
Call this file "ForLoopExample.java".
*/

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

        for(int x = 15; x < 25; x = x + 1) {
            System.out.print("value of x : " + x );
            System.out.print("\n");
        }
        }

}
```

Output



```
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
value of x : 20
value of x : 21
value of x : 22
value of x : 23
value of x : 24
Press any key to continue . . .
```

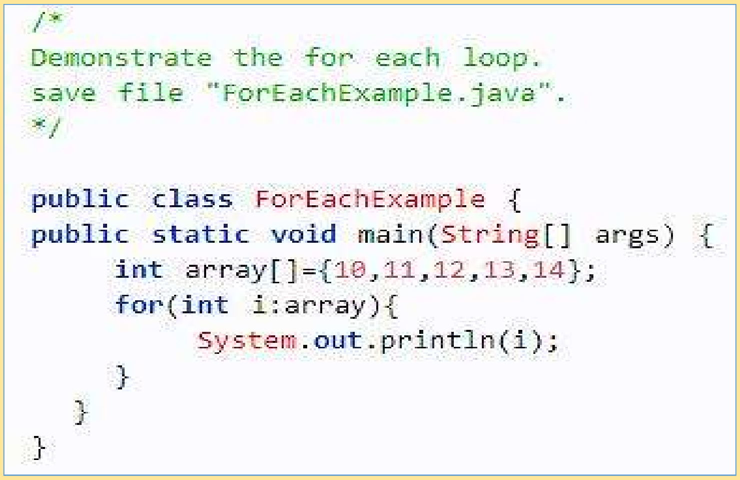
❖ For-each or Enhanced For Loop

➢ The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

Syntax

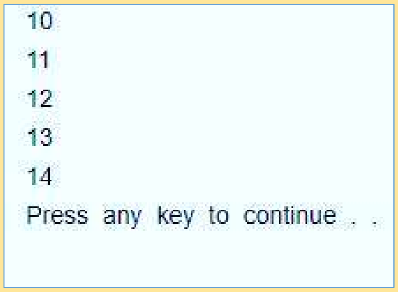
```
for (type variableName : arrayName)
 {
     // code block to be executed
 }
```

Example



```
/*
Demonstrate the for each loop.
save file "ForEachExample.java".
*/

public class ForEachExample {
public static void main(String[] args) {
      int array[]={10,11,12,13,14};
      for(int i:array){
            System.out.println(i);
      }
   }
}
```

Output



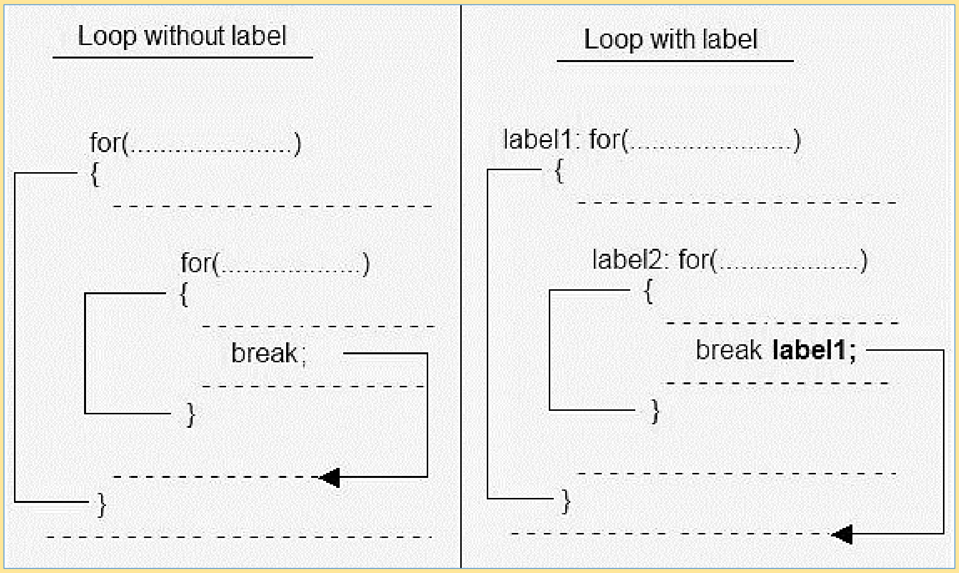
```
10
11
12
13
14
Press any key to continue . .
```

❖Labeled For Loop

- According to nested loop, if we put break statement in inner loop, compiler will jump out from inner loop and continue the outer loop again.
- What if we need to jump out from the outer loop using break statement given inside inner loop? The answer is, we should define **label** along with colon(:) sign before loop.

Syntax

```
labelname:
for(initialization; condition; increment/decrement)
 {
    //code to be executed
 }
```

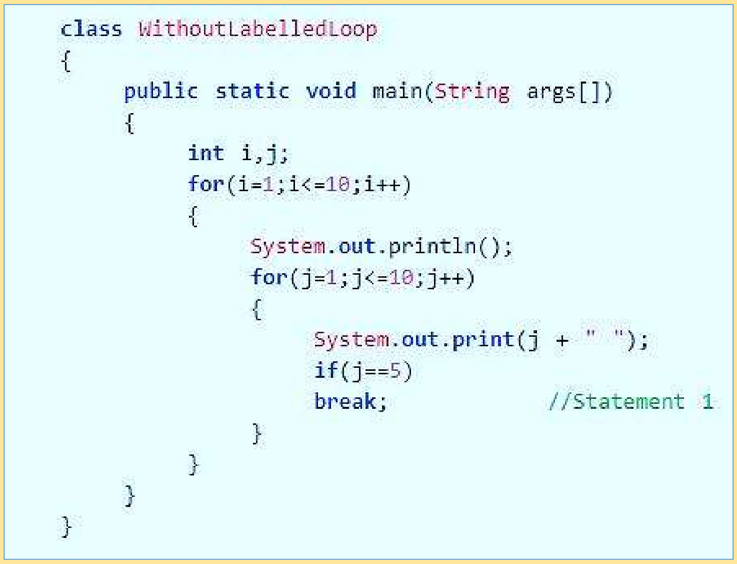


```
Loop without label

for(.....................)
{
  --------------------
     for(..................)
     {
       ---------------
       break;
       ---------------
     }
  ----------------
}
---------- --------------
```

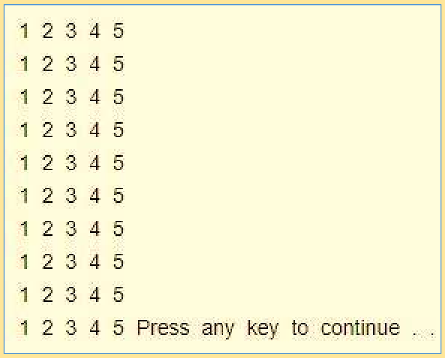
```
Loop with label

label1: for(.....................)
{
  --------------------
     label2: for(..................)
     {
       ---------------
       break label1;
       ---------------
     }
  ----------------------
}
---------- ----------
```

Example without labelled loop

Output



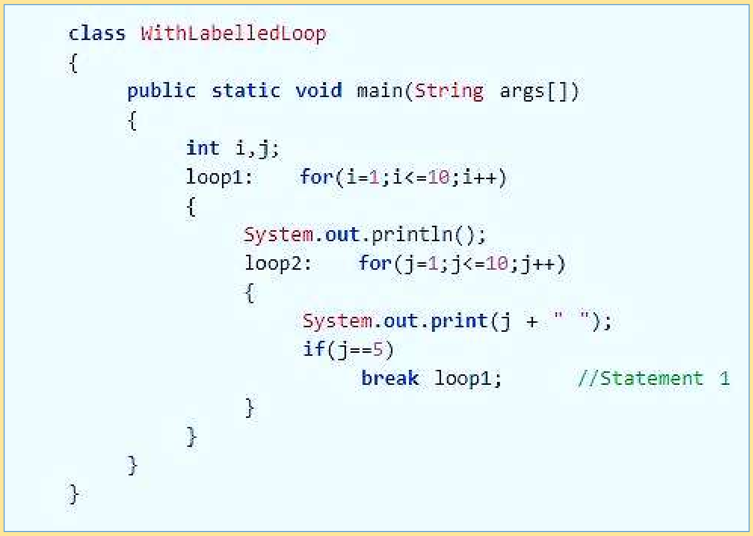
```
class WithoutLabelledLoop
{
    public static void main(String args[])
    {
        int i,j;
        for(i=1;i<=10;i++)
        {
            System.out.println();
            for(j=1;j<=10;j++)
            {
                System.out.print(j + " ");
                if(j==5)
                break;          //Statement 1
            }
        }
    }
}
```



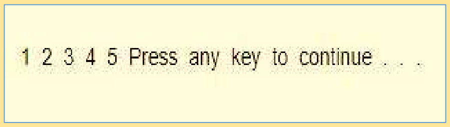
```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5 Press any key to continue . . .
```

Example with labelled loop

Output



```
class WithLabelledLoop
{
        public static void main(String args[])
        {
                int i,j;
                loop1:    for(i=1;i<=10;i++)
                {
                        System.out.println();
                        loop2:    for(j=1;j<=10;j++)
                        {
                                System.out.print(j + " ");
                                if(j==5)
                                        break loop1;      //Statement 1
                        }
                }
        }
}
```

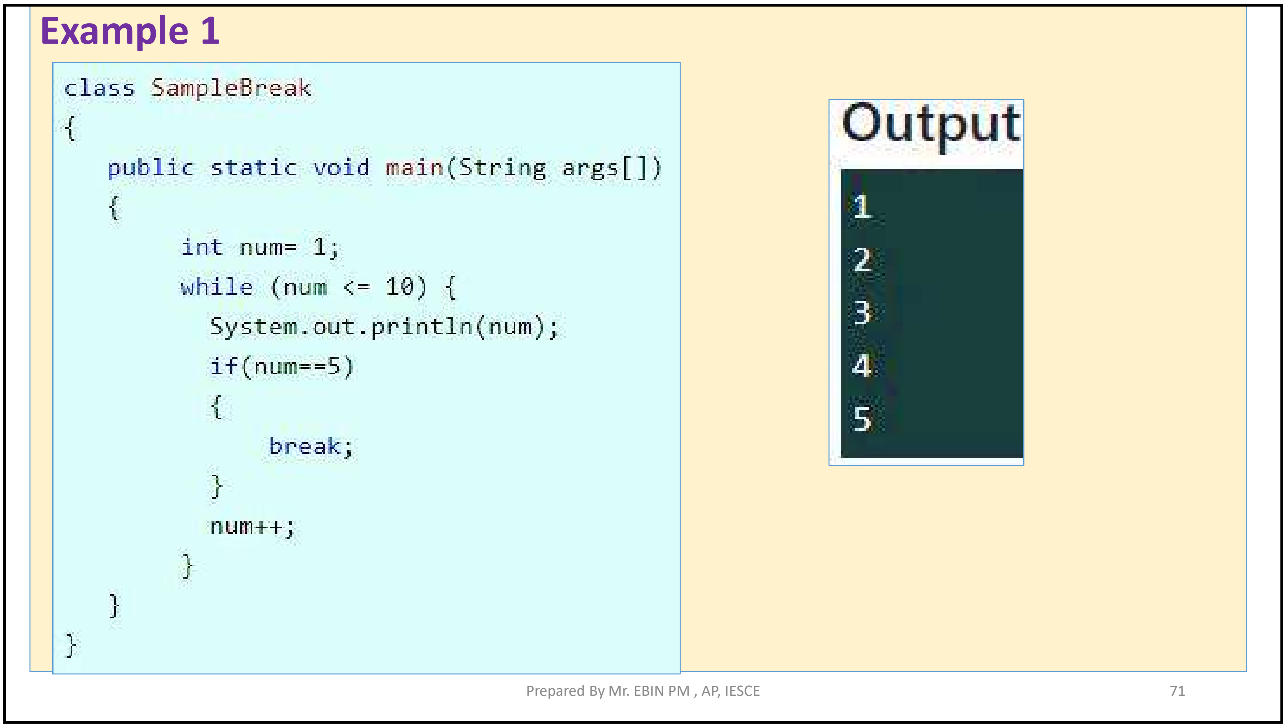


```
1 2 3 4 5 Press any key to continue . . .
```

Jump Statements

❖ Java Break Statement

- ➢The Java break statement is used to break loop or switch statement
- ➢ It breaks the current flow of the program at specified condition
- ➢When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- ➢In case of inner loop, it breaks only inner loop.

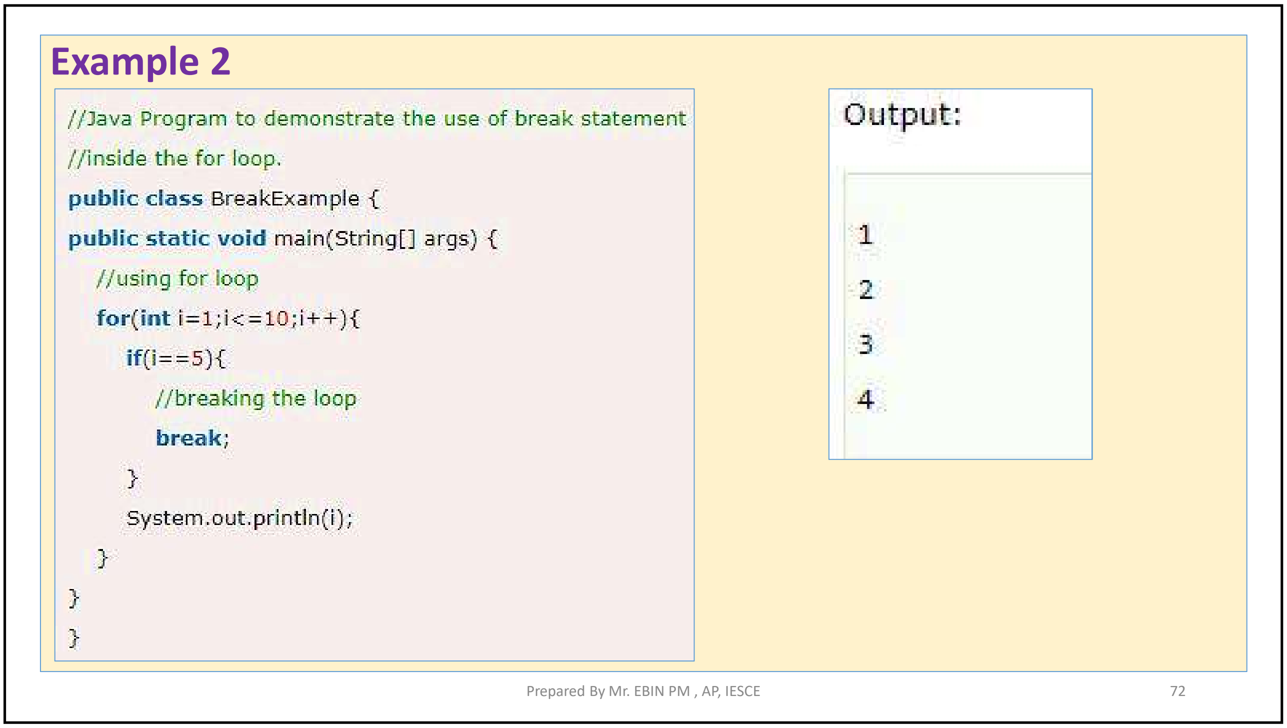


Example 1

```
class SampleBreak
{
   public static void main(String args[])
   {
        int num= 1;
        while (num <= 10) {
          System.out.println(num);
          if(num==5)
          {
              break;
          }
          num++;
        }
   }
}
```

Output

```
1
2
3
4
5
```

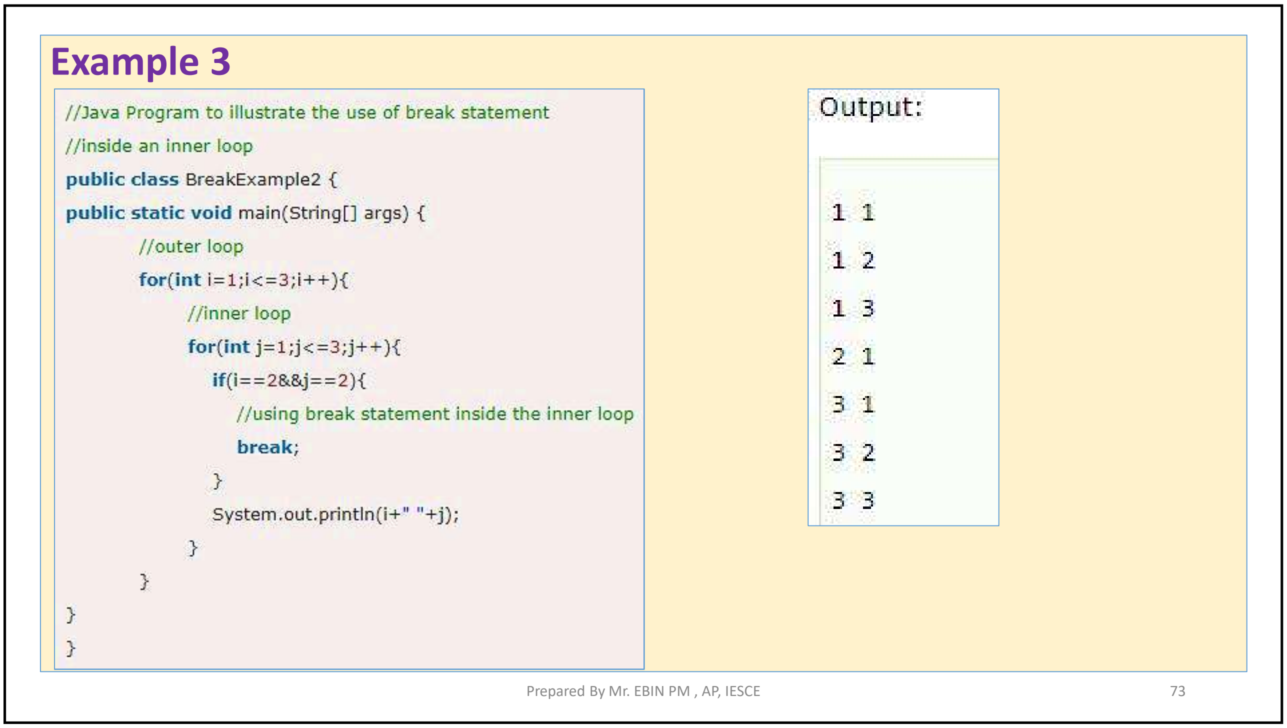


Example 2

```
//Java Program to demonstrate the use of break statement
//inside the for loop.
public class BreakExample {
public static void main(String[] args) {
    //using for loop
    for(int i=1;i<=10;i++){
        if(i==5){
            //breaking the loop
            break;
        }
        System.out.println(i);
    }
}
}
```

Output:

```
1
2
3
4
```



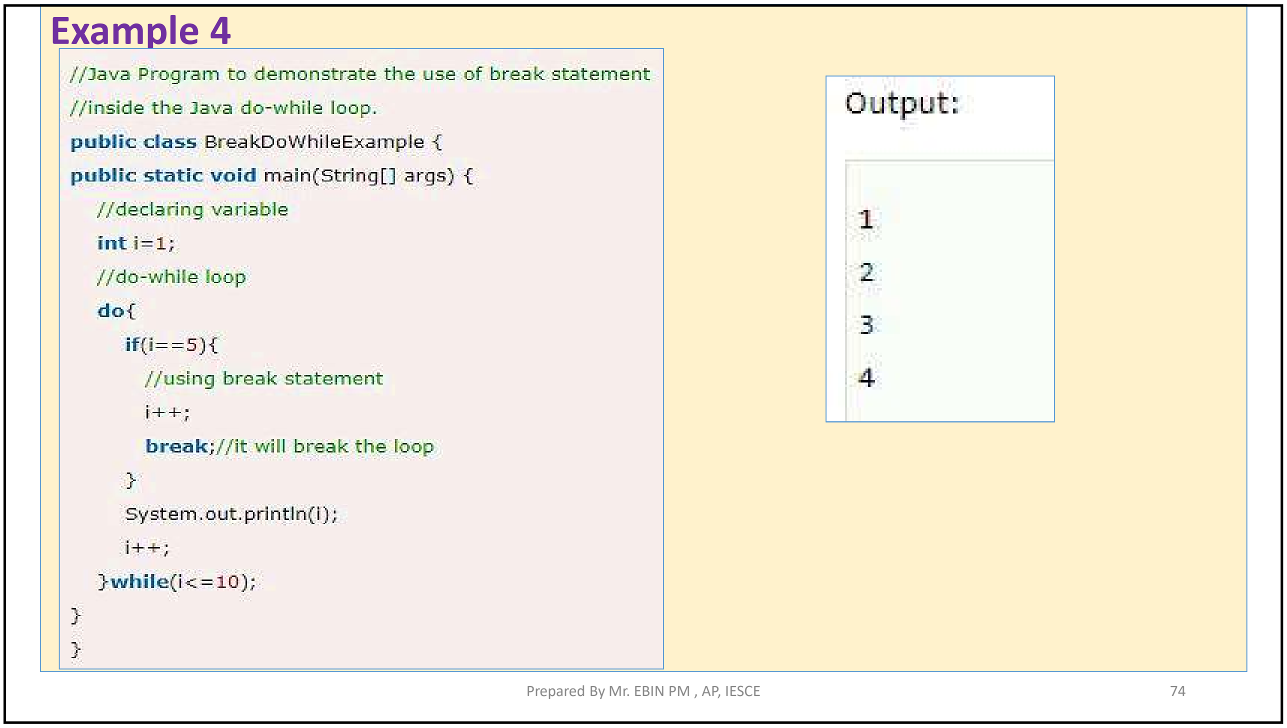
Example 3

```
//Java Program to illustrate the use of break statement
//inside an inner loop
public class BreakExample2 {
public static void main(String[] args) {
        //outer loop
        for(int i=1;i<=3;i++){
            //inner loop
            for(int j=1;j<=3;j++){
              if(i==2&&j==2){
                //using break statement inside the inner loop
                break;
              }
              System.out.println(i+" "+j);
            }
        }
}
}
```

Output:

```
1 1
1 2
1 3
2 1
3 1
3 2
3 3
```

Prepared By Mr. EBIN PM , AP, IESCE 73



Example 4

```
//Java Program to demonstrate the use of break statement
//inside the Java do-while loop.
public class BreakDoWhileExample {
public static void main(String[] args) {
  //declaring variable
  int i=1;
  //do-while loop
  do{
    if(i==5){
      //using break statement
      i++;
      break;//it will break the loop
    }
    System.out.println(i);
    i++;
  }while(i<=10);
}
}
```

Output:

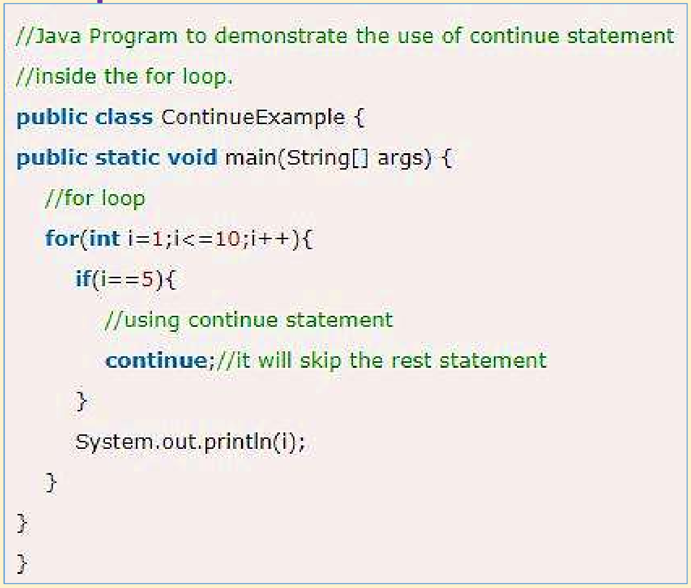
```
1
2
3
4
```

Prepared By Mr. EBIN PM , AP, IESCE 74

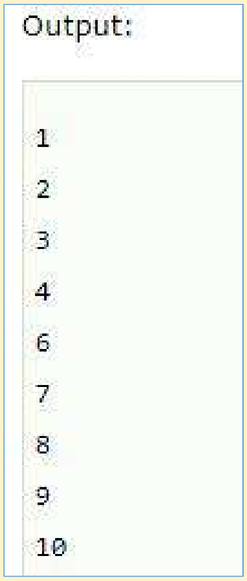
❖ Java Continue Statement

- The Java continue statement is used to continue the loop
- The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately
- It continues the current flow of the program and skips the remaining code at the specified condition.
- In case of an inner loop, it continues the inner loop only.

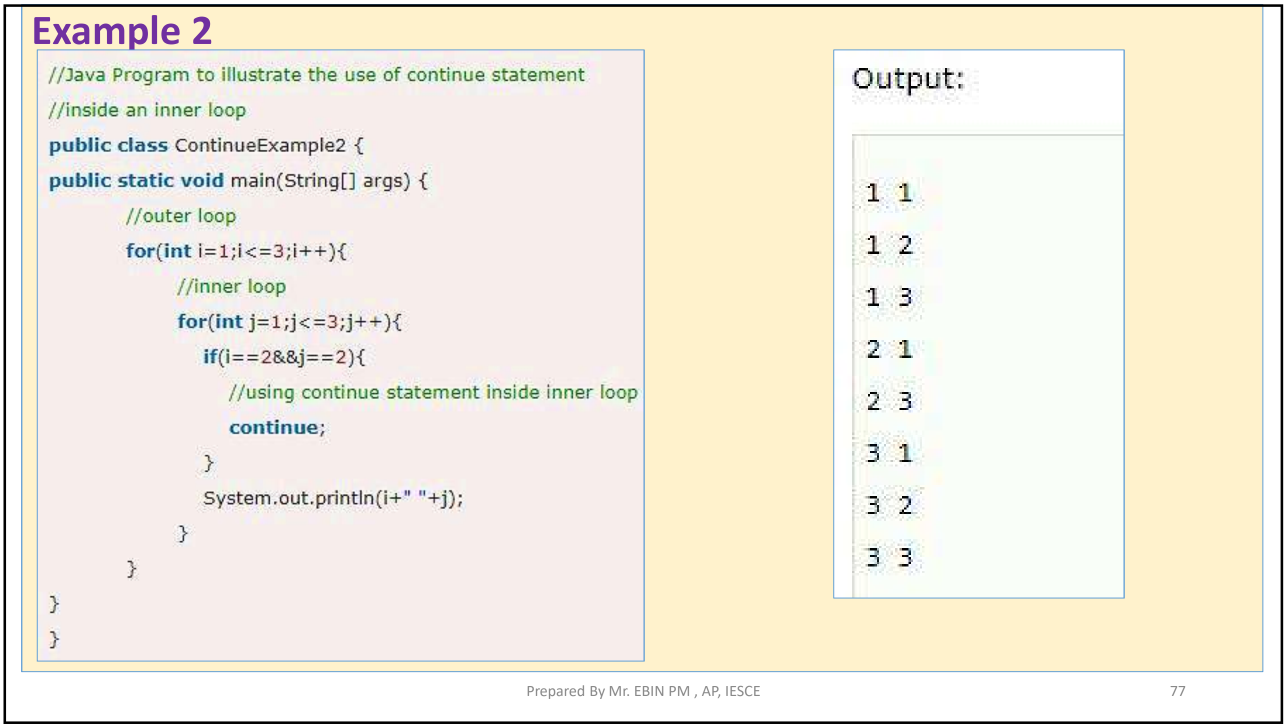
Example 1



```
//Java Program to demonstrate the use of continue statement
//inside the for loop.
public class ContinueExample {
public static void main(String[] args) {
    //for loop
    for(int i=1;i<=10;i++){
        if(i==5){
            //using continue statement
            continue;//it will skip the rest statement
        }
        System.out.println(i);
    }
}
}
```



```
Output:

1
2
3
4
6
7
8
9
10
```



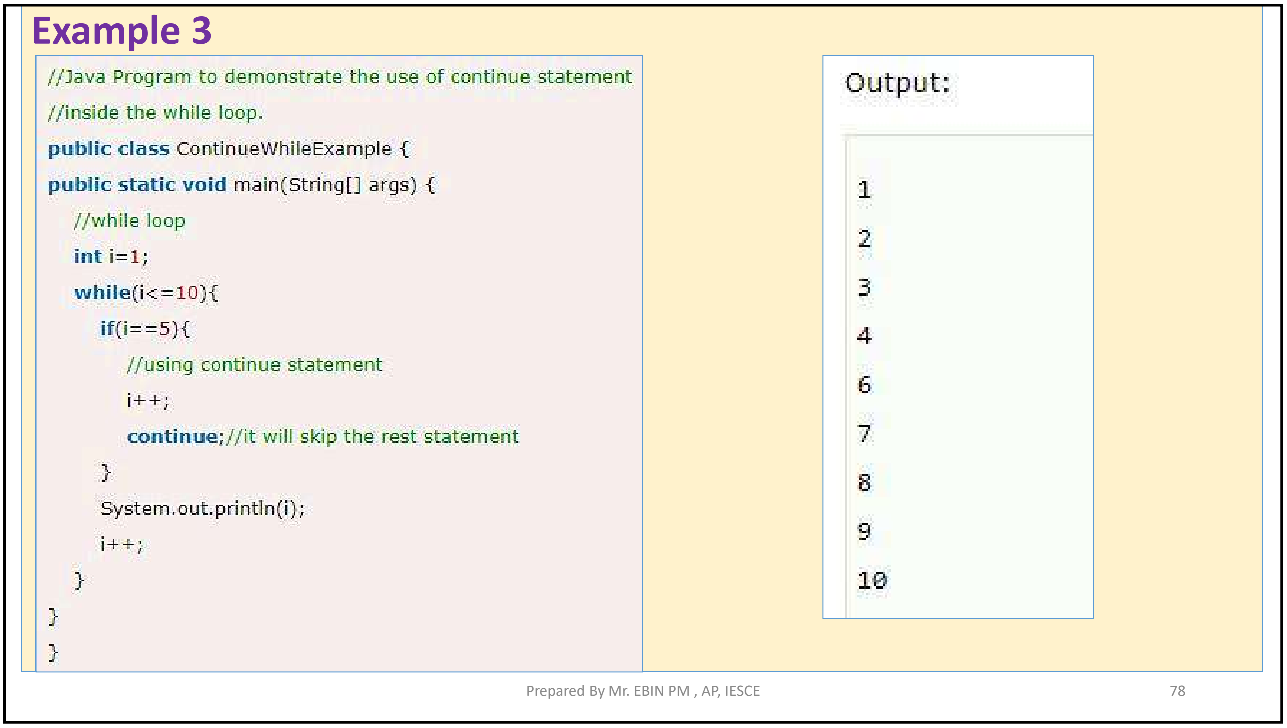
Example 2

```
//Java Program to illustrate the use of continue statement
//inside an inner loop
public class ContinueExample2 {
public static void main(String[] args) {
            //outer loop
            for(int i=1;i<=3;i++){
                    //inner loop
                    for(int j=1;j<=3;j++){
                        if(i==2&&j==2){
                            //using continue statement inside inner loop
                            continue;
                        }
                        System.out.println(i+" "+j);
                    }
            }
}
}
```

Output:

```
1 1
1 2
1 3
2 1
2 3
3 1
3 2
3 3
```

Prepared By Mr. EBIN PM , AP, IESCE 77



Example 3

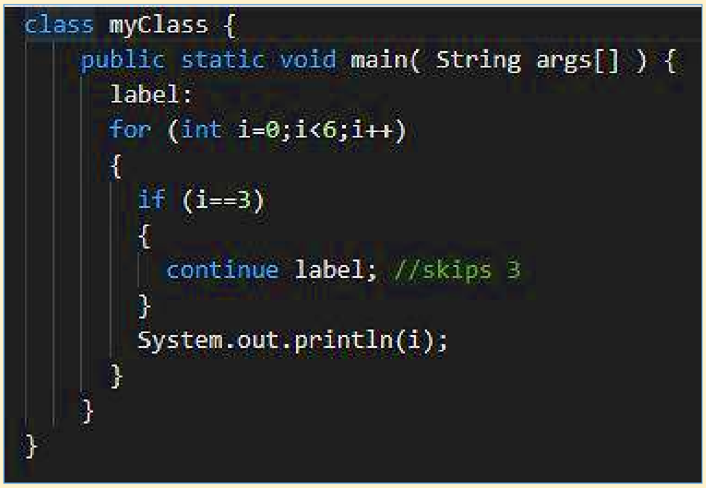
```
//Java Program to demonstrate the use of continue statement
//inside the while loop.
public class ContinueWhileExample {
public static void main(String[] args) {
    //while loop
    int i=1;
    while(i<=10){
        if(i==5){
            //using continue statement
            i++;
            continue;//it will skip the rest statement
        }
        System.out.println(i);
        i++;
    }
}
}
```

Output:

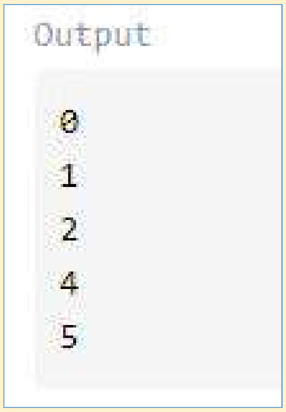
```
1
2
3
4
6
7
8
9
10
```

Prepared By Mr. EBIN PM , AP, IESCE 78

Example 4



```
class myClass {
    public static void main( String args[] ) {
      label:
      for (int i=0;i<6;i++)
      {
        if (i==3)
        {
          continue label; //skips 3
        }
        System.out.println(i);
      }
    }
}
```



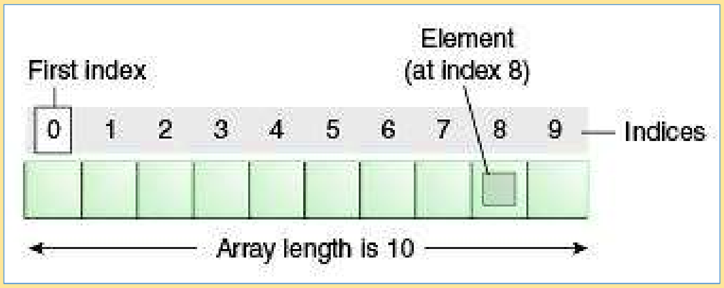
Output

```
0
1
2
4
5
```

ARRAY

- An array is a collection of similar data types.
- **Java array** is an object which contains elements of a similar data type.
- The elements of an array are stored in a contiguous memory location
- the size of an array is fixed and cannot increase to accommodate more elements
- It is also known as static data structure because size of an array must be specified at the time of its declaration.
- Array in Java is index-based, the first element of the array is stored at the 0th index

- Java provides the feature of anonymous arrays which is not available in C/C++.



Advantage of Java Array

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data easily.
- **Random access:** We can get any data located at any index position.

Disadvantage of Java Array

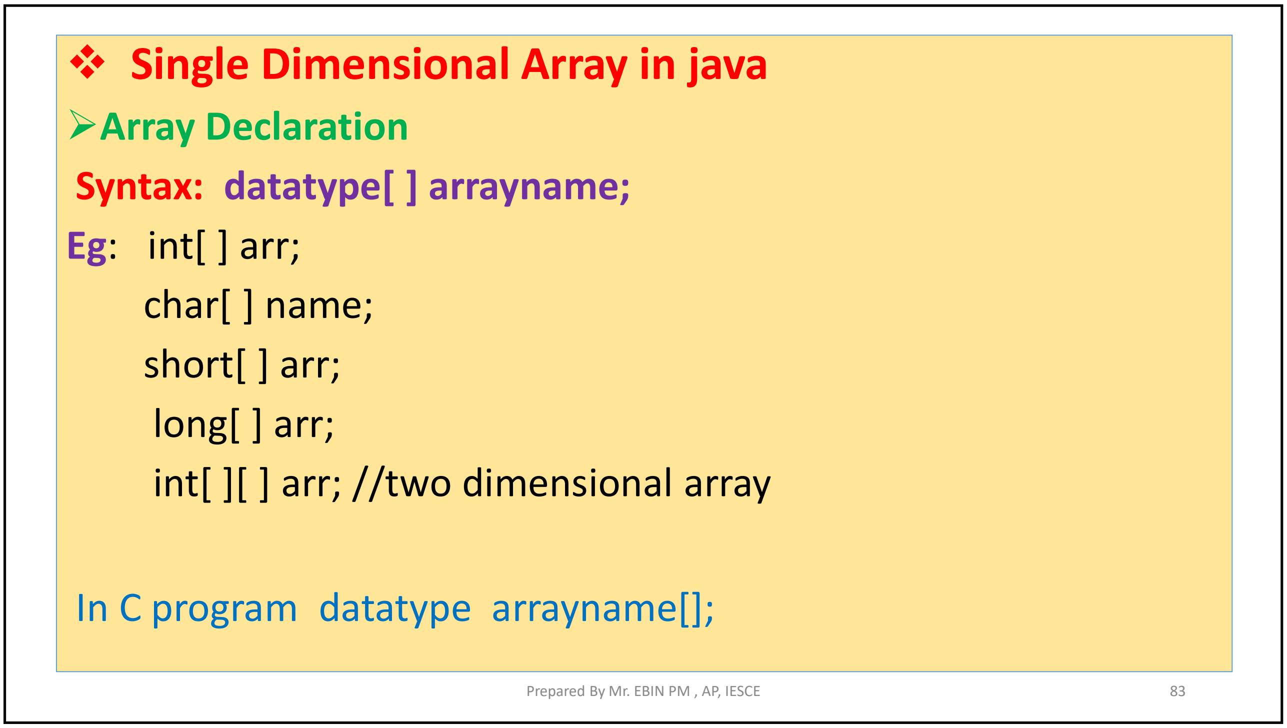
- **Size Limit:** We can store the only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

Features of Array

- It is always indexed. The index begins from 0.
- It is a collection of similar data types.
- It occupies a contiguous memory location.

Types of Java Array

- Single Dimensional Array
- Multidimensional Array



❖ Single Dimensional Array in java

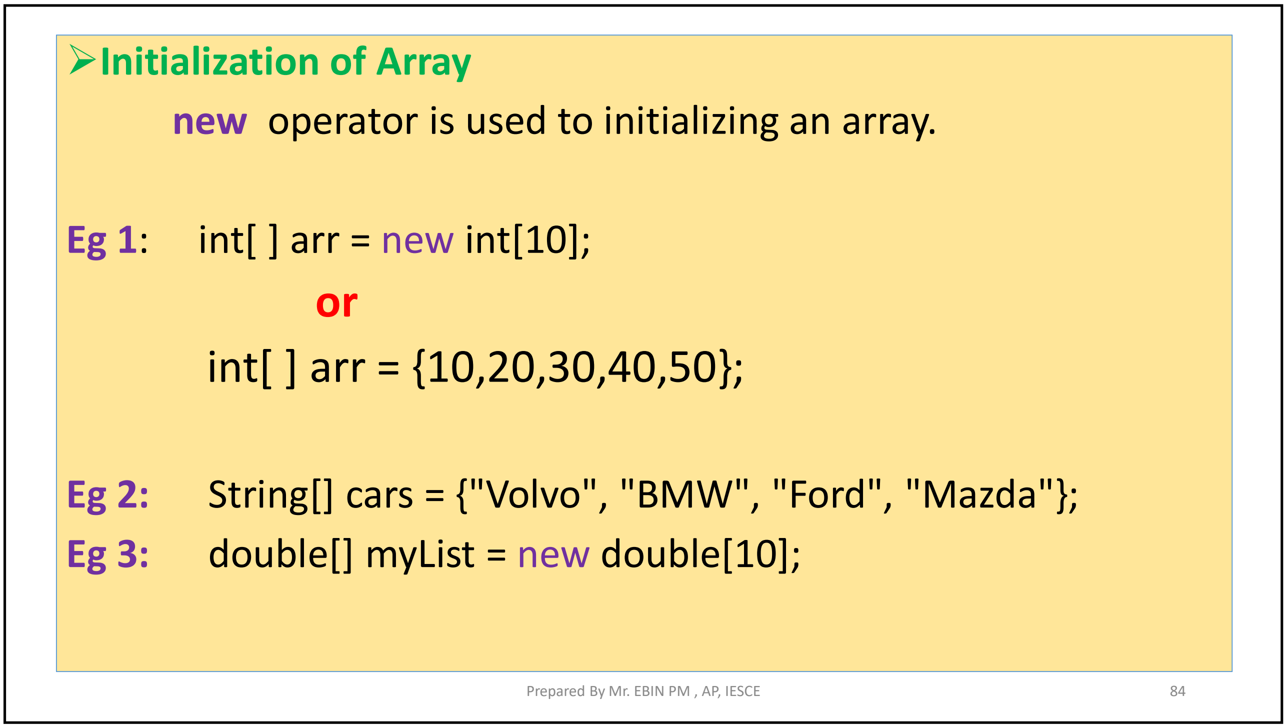
➢Array Declaration

Syntax: datatype[ ] arrayname;

Eg:

```
int[ ] arr;
char[ ] name;
short[ ] arr;
 long[ ] arr;
 int[ ][ ] arr; //two dimensional array
```

In C program datatype arrayname[];



➢Initialization of Array

new operator is used to initializing an array.

Eg 1:

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

or

```
int[ ] arr = {10,20,30,40,50};
```

Eg 2:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

Eg 3:

```
double[] myList = new double[10];
```

➢Accessing array element

Example: To access 4th element of a given array

```
        int[ ] arr = {10,24,30,50};
        System.out.println("Element at 4th place" + arr[3]);
```

➢To find the length of an array, we can use the following syntax: array_name.length

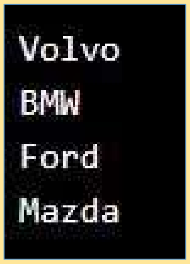
Example:

```
public class MyClass
 {
   public static void main(String[] args)
    {
       String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
       System.out.println(cars.length);
    }
 }
```

Output 4

➢Loop Through an Array

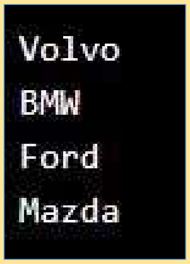
```
public class MyClass
{
  public static void main(String[] args)
     {
        String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
        for (int i = 0; i < cars.length; i++)
          {
             System.out.println(cars[i]);
          }
     }
}
```



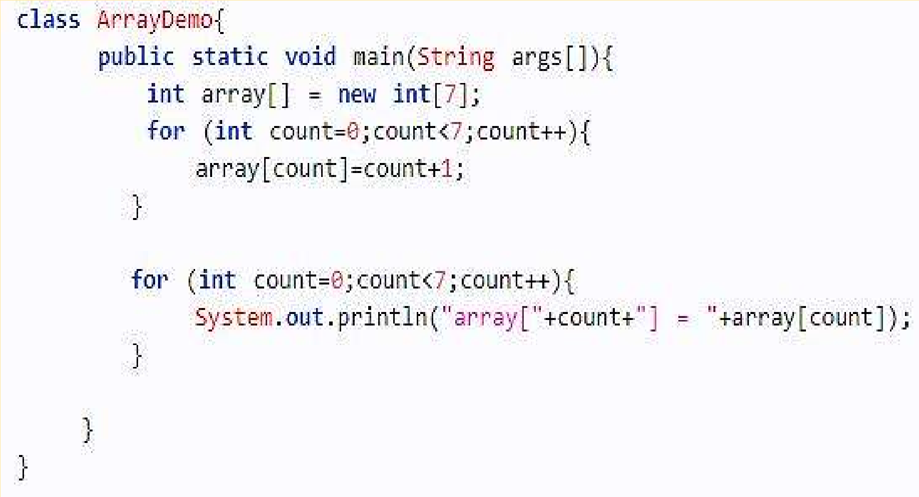
```
Volvo
BMW
Ford
Mazda
```

➢Loop Through an Array with For-Each

```
public class MyClass
{
   public static void main(String[] args)
      {
        String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
        for (String i : cars)
          {
              System.out.println(i);
          }
      }
}
```

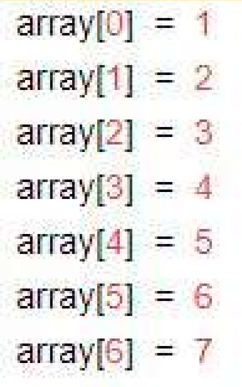


```
Volvo
BMW
Ford
Mazda
```

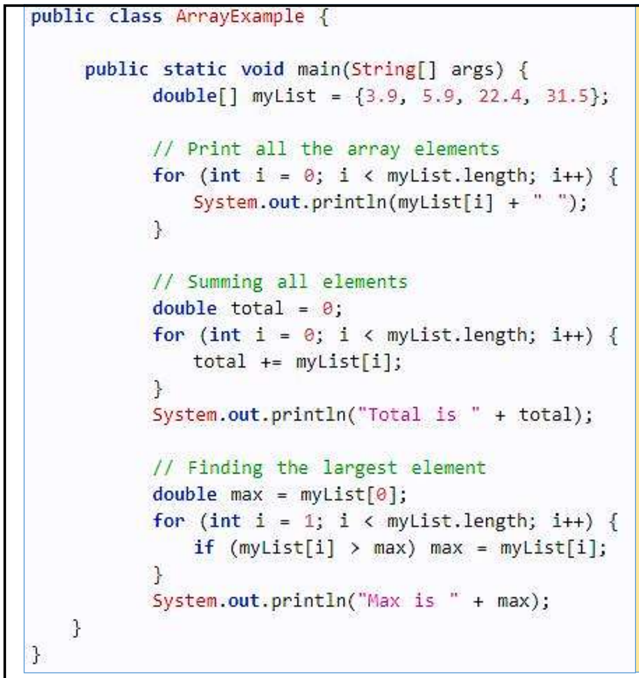


```
class ArrayDemo{
      public static void main(String args[]){
         int array[] = new int[7];
         for (int count=0;count<7;count++){
            array[count]=count+1;
        }

        for (int count=0;count<7;count++){
            System.out.println("array["+count+"] = "+array[count]);
        }

    }
}
```

Output

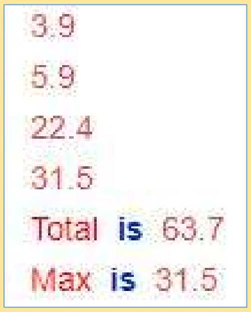


```
array[0] = 1
array[1] = 2
array[2] = 3
array[3] = 4
array[4] = 5
array[5] = 6
array[6] = 7
```



```
public class ArrayExample {

    public static void main(String[] args) {
        double[] myList = {3.9, 5.9, 22.4, 31.5};

        // Print all the array elements
        for (int i = 0; i < myList.length; i++) {
            System.out.println(myList[i] + " ");
        }

        // Summing all elements
        double total = 0;
        for (int i = 0; i < myList.length; i++) {
            total += myList[i];
        }
        System.out.println("Total is " + total);

        // Finding the largest element
        double max = myList[0];
        for (int i = 1; i < myList.length; i++) {
            if (myList[i] > max) max = myList[i];
        }
        System.out.println("Max is " + max);
    }
}
```

Output



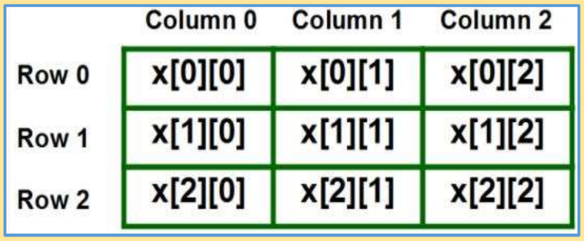
```
3.9
5.9
22.4
31.5
Total is 63.7
Max is 31.5
```

❖Two Dimensional array

➢**Array Declaration**

Syntax : datatype[ ][ ] arrayname;

Eg: int[][] myNumbers ;



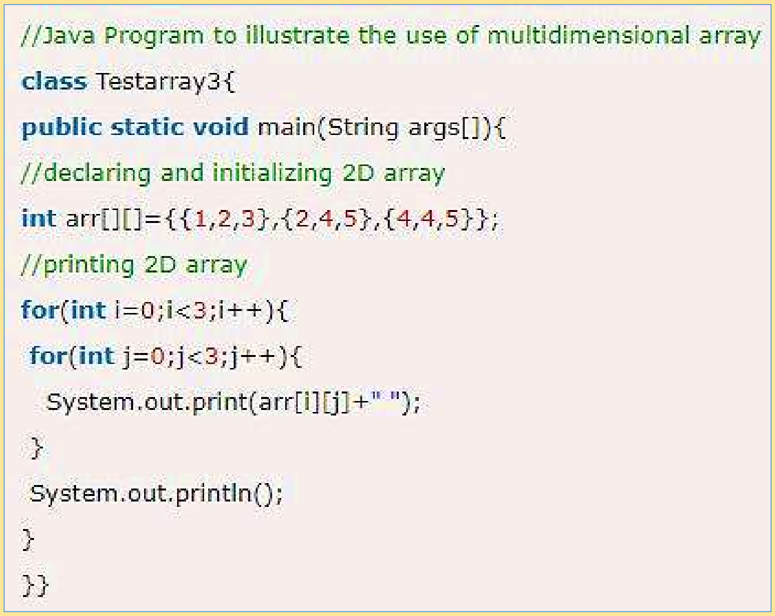
	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

➢**Array Initialization**

int[ ][ ] arrName = new int[10][10];

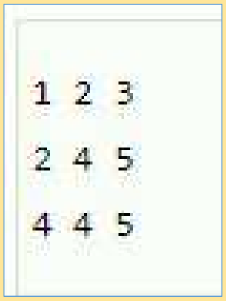
Or

int[ ][ ] arrName = {{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}}; // 3 by 5 is the size of the array.

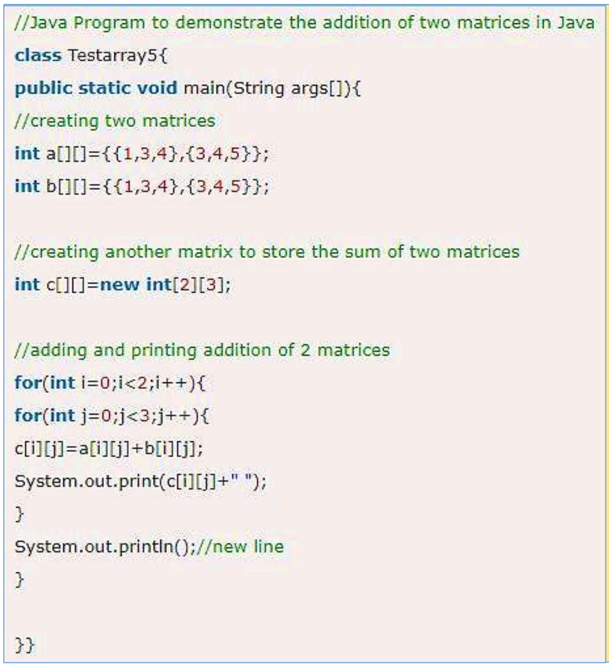


```
//Java Program to illustrate the use of multidimensional array
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
}}
```

Output

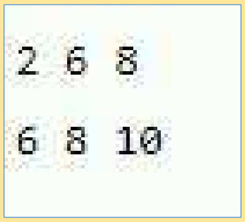


```
1 2 3
2 4 5
4 4 5
```



```
//Java Program to demonstrate the addition of two matrices in Java
class Testarray5{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};

//creating another matrix to store the sum of two matrices
int c[][]=new int[2][3];

//adding and printing addition of 2 matrices
for(int i=0;i<2;i++){
for(int j=0;j<3;j++){
c[i][j]=a[i][j]+b[i][j];
System.out.print(c[i][j]+" ");
}
System.out.println();//new line
}

}}
```

Output



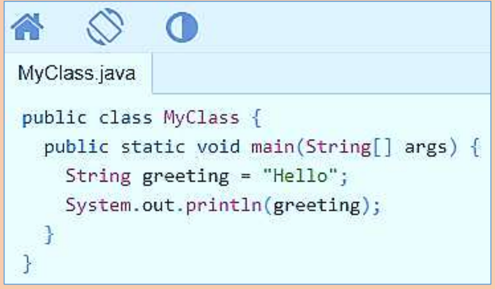
```
2 6 8
6 8 10
```

STRING

- Strings are used for storing text
- A String variable contains a collection of characters surrounded by double quotes

Eg: Create a variable of type String and assign it a value

String greeting = "Hello";



MyClass.java

```
public class MyClass {
  public static void main(String[] args) {
    String greeting = "Hello";
    System.out.println(greeting);
  }
}
```

Output

```
Hello
```

- In Java, string is basically an object that represents sequence of char values
- An array of characters works same as Java string. For example:

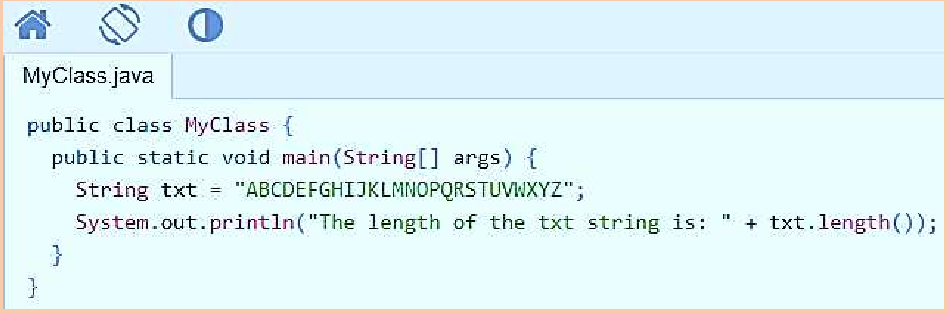
```
char[] ch={'j',‘o',‘s',‘e',‘p',‘h'};
String s=new String(ch);   //converting char array to string
```

is same as

```
String s="joseph";     //creating string by java string literal
```

❖String Length

- The length of a string can be found with the length() method

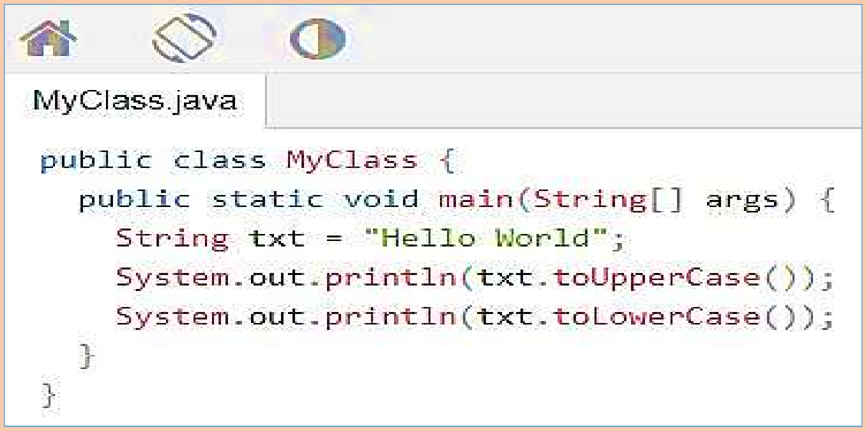


```
public class MyClass {
  public static void main(String[] args) {
    String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    System.out.println("The length of the txt string is: " + txt.length());
  }
}
```

Output

The length of the txt string is: 26

❖toUpperCase() and toLowerCase()



```
public class MyClass {
  public static void main(String[] args) {
    String txt = "Hello World";
    System.out.println(txt.toUpperCase());
    System.out.println(txt.toLowerCase());
  }
}
```

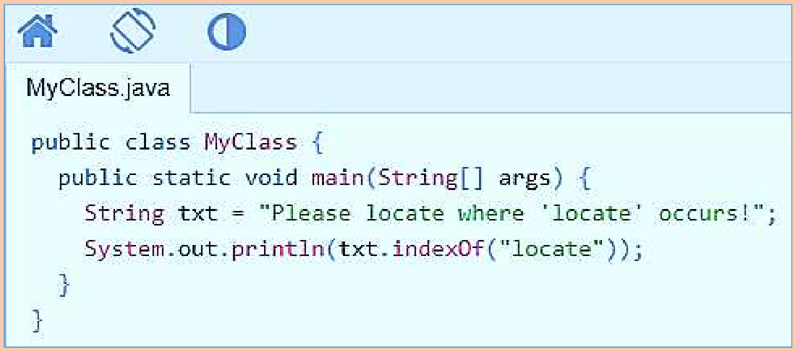
Output

HELLO WORLD

hello world

❖Finding a Character in a String

➢The indexOf() method returns the index (the position) of the first occurrence of a specified text in a string (including whitespace)

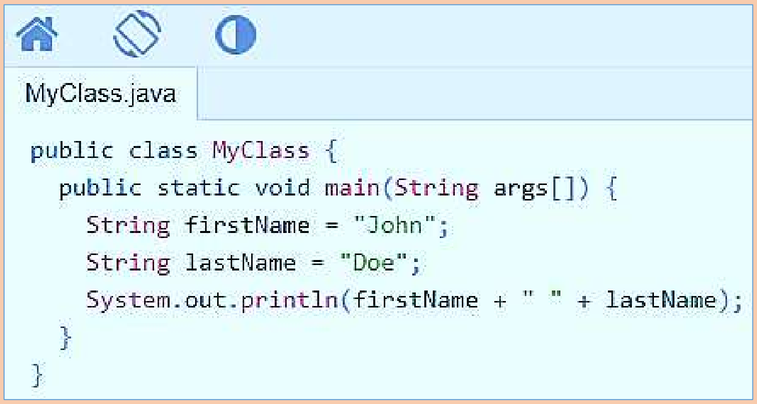


```
public class MyClass {
  public static void main(String[] args) {
    String txt = "Please locate where 'locate' occurs!";
    System.out.println(txt.indexOf("locate"));
  }
}
```

Output **7**

❖String Concatenation

- The + operator can be used between strings to combine them. This is called concatenation

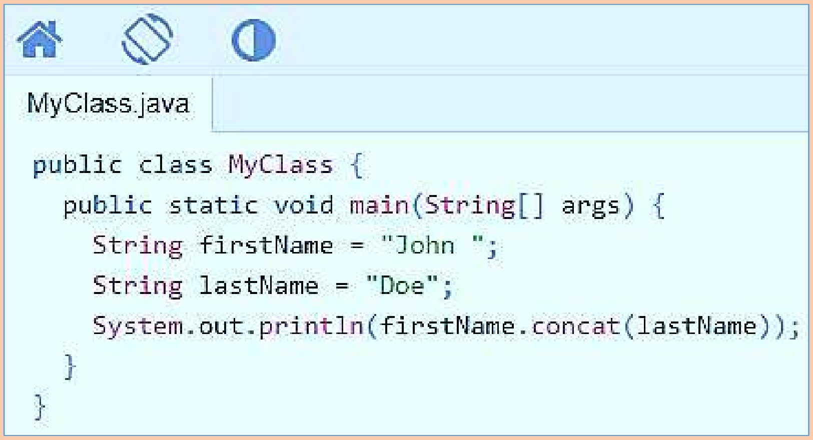


```
public class MyClass {
  public static void main(String args[]) {
    String firstName = "John";
    String lastName = "Doe";
    System.out.println(firstName + " " + lastName);
  }
}
```

Output John Doe

❖concat() method

- We can also use the concat() method to concatenate two strings:



```
public class MyClass {
  public static void main(String[] args) {
    String firstName = "John ";
    String lastName = "Doe";
    System.out.println(firstName.concat(lastName));
  }
}
```

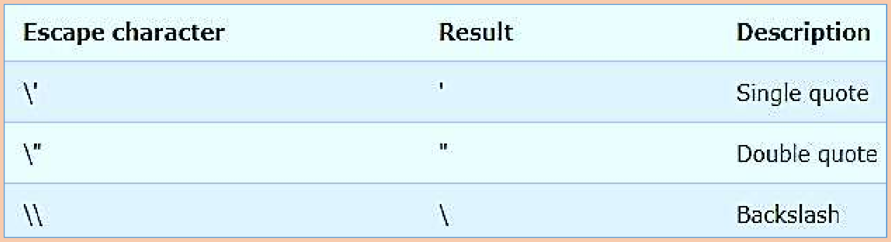
Output John Doe

❖Special Characters

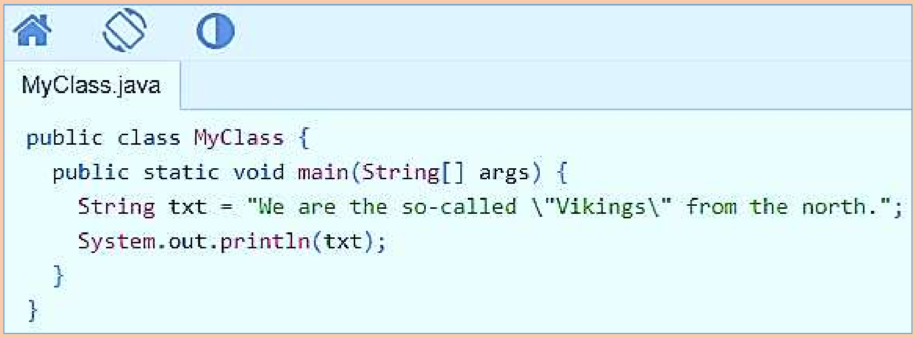
Consider the following example

String txt = "We are the so-called "Vikings" from the north.";

- Because strings must be written within quotes, Java will misunderstand this string
- The solution to avoid this problem, is to use the backslash escape character



Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash



MyClass.java

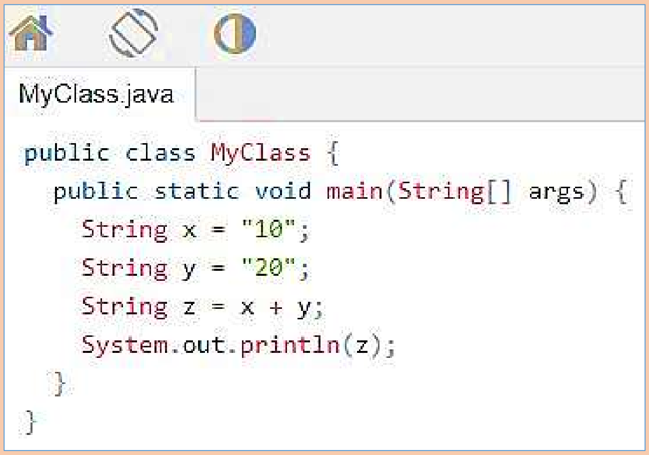
```
public class MyClass {
  public static void main(String[] args) {
    String txt = "We are the so-called \"Vikings\" from the north.";
    System.out.println(txt);
  }
}
```

Output We are the so-called "Vikings" from the north.

- The sequence \" inserts a double quote in a string
- The sequence \' inserts a single quote in a string
- The sequence \\ inserts a single backslash in a string

❖Adding Numbers and Strings

- Java uses the + operator for both addition and concatenation.
- If we add two strings, the result will be a string concatenation



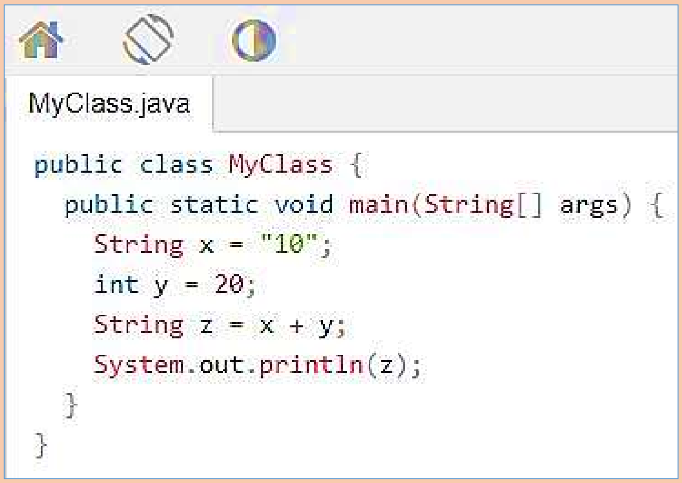
MyClass.java

```
public class MyClass {
  public static void main(String[] args) {
    String x = "10";
    String y = "20";
    String z = x + y;
    System.out.println(z);
  }
}
```

Output

```
1020
```

- If we add a number and a string, the result will be a string concatenation



```
public class MyClass {
  public static void main(String[] args) {
    String x = "10";
    int y = 20;
    String z = x + y;
    System.out.println(z);
  }
}
```

Output

1020