JAVA

UNIT 3: OPERATORS AND CONTROL STATEMENTS

Operators:

- ✓ **Operator** in Java is a symbol which is used to perform operations.
- ✓ For example: +, -, *, / etc.
- ✓ There are **many types of operators** in Java which are given below:
    - Arithmetic Operator
    - Relational Operator
    - Logical Operator
    - Ternary/Conditional Operator
    - Assignment Operator
    - Bitwise Operator

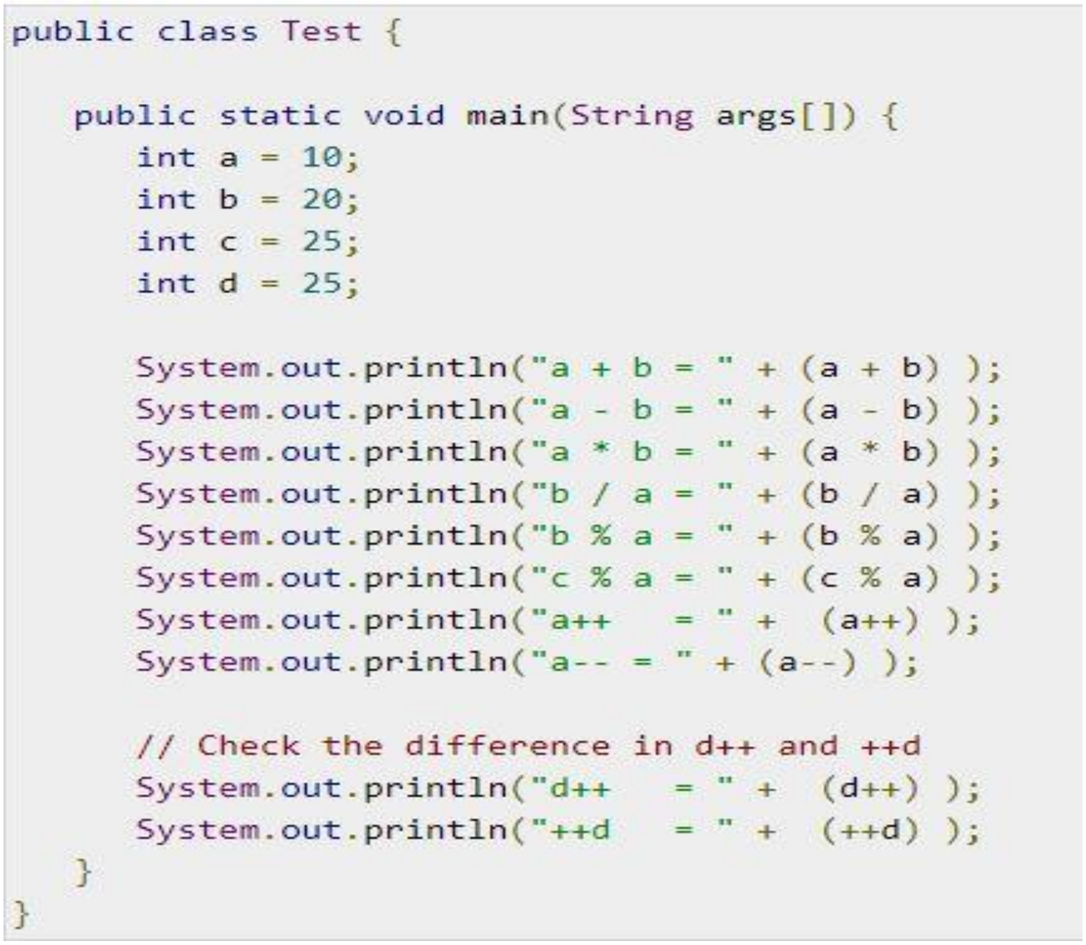
Arithmetic Operator:

- Arithmetic operators are used in mathematical expressions.
- Assume **integer variable A=10** and **variable B=20**, then –

Operator	Example
+ (Addition)	A + B = 30
- (Subtraction)	A - B = -10
* (Multiplication)	A * B = 200
/ (Division)	B / A = 2
% (Modulus)	B % A = 0
++ (Increment)	B++ gives 21

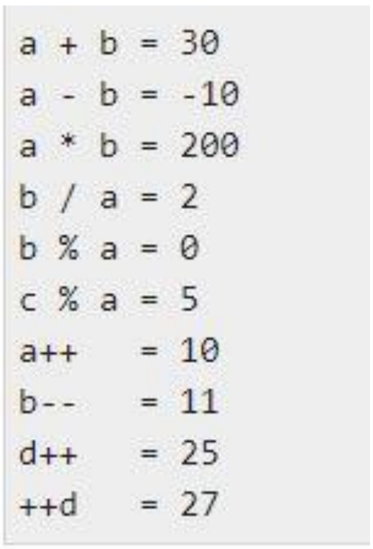
-- (Decrement)	B-- gives 19

Example:

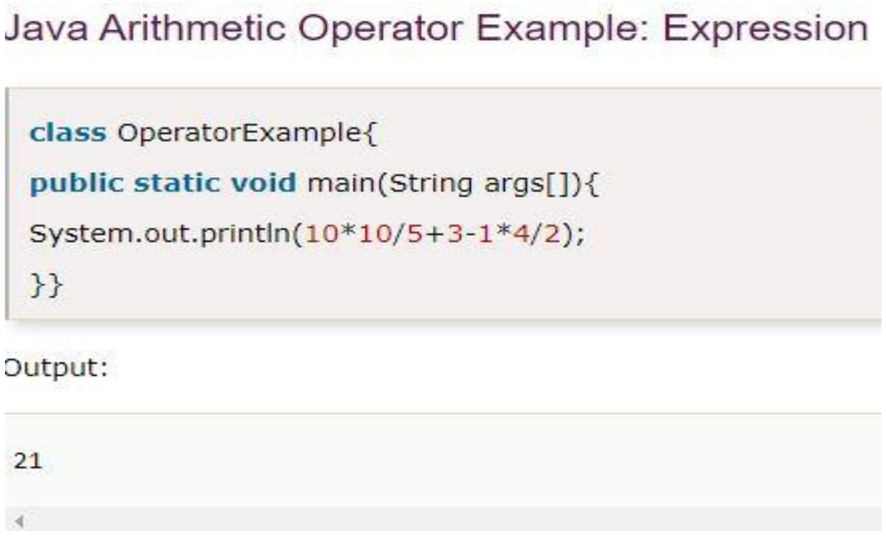


```
public class Test {

   public static void main(String args[]) {
      int a = 10;
      int b = 20;
      int c = 25;
      int d = 25;

      System.out.println("a + b = " + (a + b) );
      System.out.println("a - b = " + (a - b) );
      System.out.println("a * b = " + (a * b) );
      System.out.println("b / a = " + (b / a) );
      System.out.println("b % a = " + (b % a) );
      System.out.println("c % a = " + (c % a) );
      System.out.println("a++   = " +  (a++) );
      System.out.println("a-- = " + (a--) );

      // Check the difference in d++ and ++d
      System.out.println("d++   = " +  (d++) );
      System.out.println("++d   = " +  (++d) );
   }
}
```

Output:



```
a + b = 30
a - b = -10
a * b = 200
b / a = 2
b % a = 0
c % a = 5
a++   = 10
b--   = 11
d++   = 25
++d   = 27
```



Java Arithmetic Operator Example: Expression

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10*10/5+3-1*4/2);
}}
```

Output:

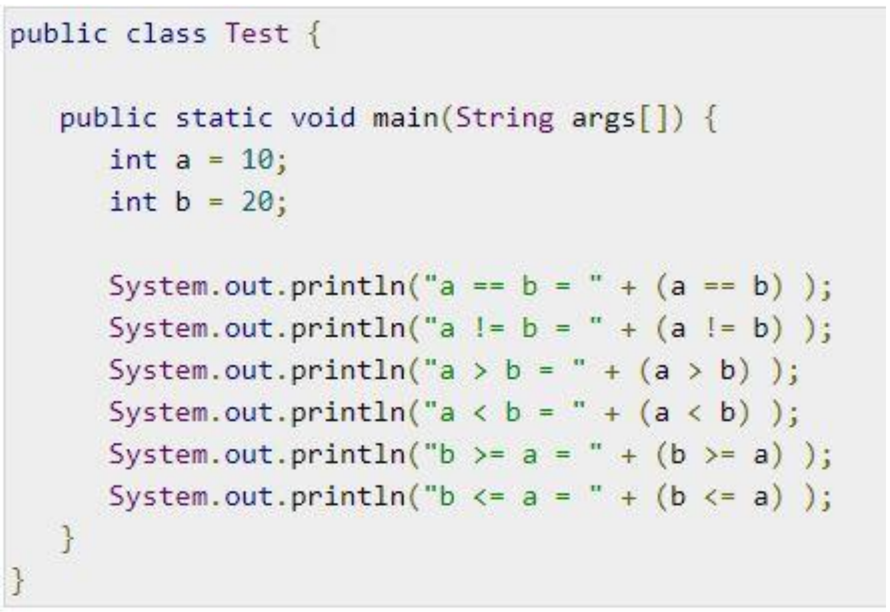
```
21
```

Relational Operator:

- Assume **variable** A = 10 and **variable** B = 20, then –

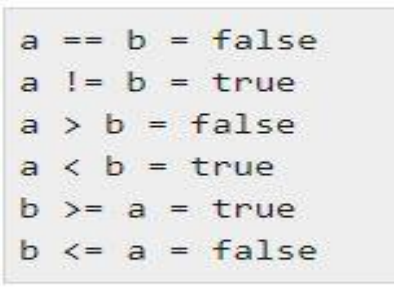
Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true else not true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A! = B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Example:



```
public class Test {

   public static void main(String args[]) {
      int a = 10;
      int b = 20;

      System.out.println("a == b = " + (a == b) );
      System.out.println("a != b = " + (a != b) );
      System.out.println("a > b = " + (a > b) );
      System.out.println("a < b = " + (a < b) );
      System.out.println("b >= a = " + (b >= a) );
      System.out.println("b <= a = " + (b <= a) );
   }
}
```

Output:



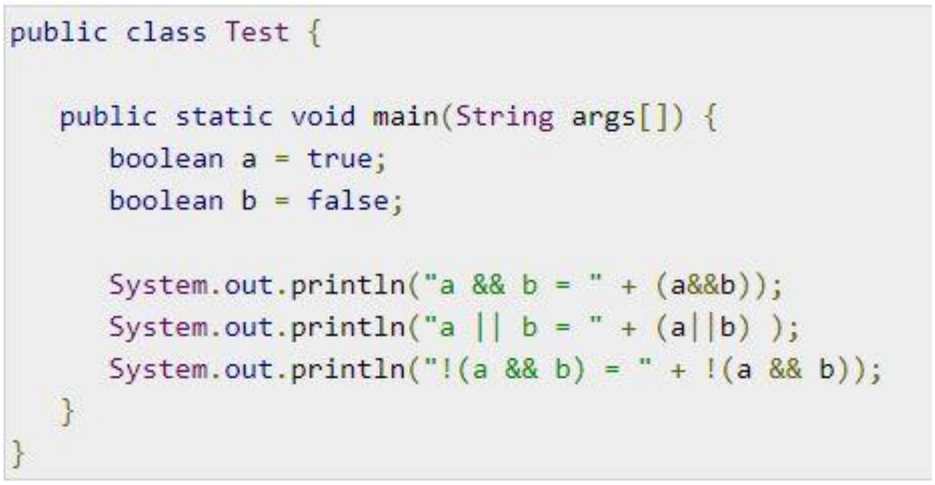
```
a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false
```

Logical Operator:

- Assume Boolean **variables** **A** **holds** **TRUE** and **variable** **B** **holds** **FALSE**, then –

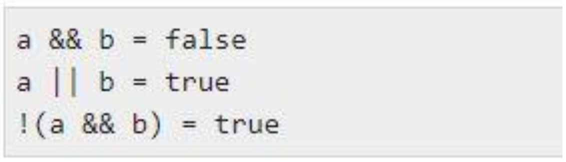
Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical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

Example:



```
public class Test {

   public static void main(String args[]) {
      boolean a = true;
      boolean b = false;

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

Output:

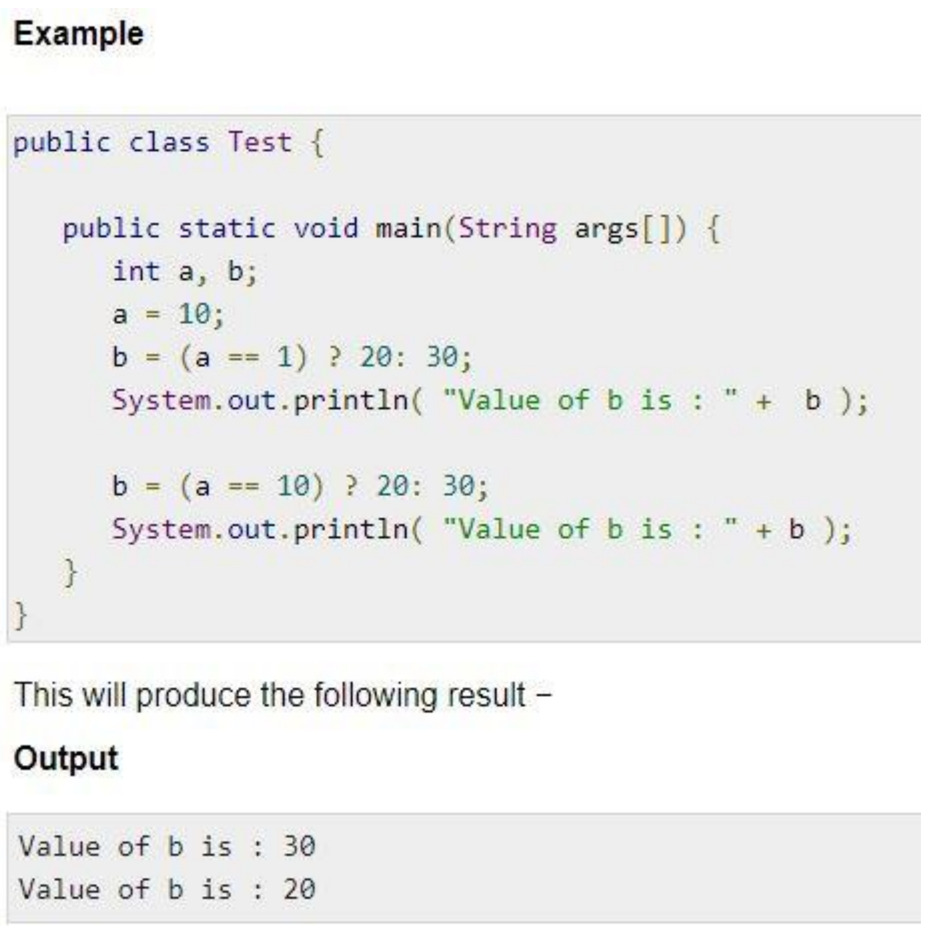


```
a && b = false
a || b = true
!(a && b) = true
```

Conditional Operator:

- Conditional operator is also known as the **ternary operator**.
- Conditional operator operator consists of **three operands** and is used to evaluate Boolean expressions.
- The goal of the operator is to decide, which value should be assigned to the variable.
- The operator is written as –

variable x = (expression) ? value if true : value if false



Example

```
public class Test {

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

      b = (a == 10) ? 20: 30;
      System.out.println( "Value of b is : " + b );
   }
}
```

This will produce the following result –

Output

```
Value of b is : 30
Value of b is : 20
```

Assignment Operator:

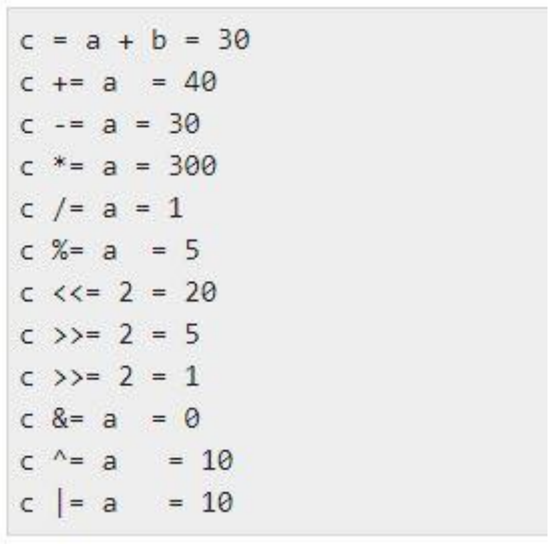
Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2

^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Example:

```
public class Test {

  public static void main(String args[]) {
    int a = 10;
    int b = 20;
    int c = 0;

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

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

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

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

    a = 10;
    c = 15;
    c /= a ;
    System.out.println("c /= a = " + c );

    a = 10;
    c = 15;
    c %= a ;
    System.out.println("c %= a  = " + c );

    c <<= 2 ;
    System.out.println("c <<= 2 = " + c );

    c >>= 2 ;
    System.out.println("c >>= 2 = " + c );

    c >>= 2 ;
    System.out.println("c >>= 2 = " + c );
    c &= a ;
    System.out.println("c &= a  = " + c );

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

    c |= a ;
    System.out.println("c |= a   = " + c );
  }
}
```

Output:



```
c = a + b = 30
c += a  = 40
c -= a = 30
c *= a = 300
c /= a = 1
c %= a  = 5
c <<= 2 = 20
c >>= 2 = 5
c >>= 2 = 1
c &= a  = 0
c ^= a   = 10
c |= a   = 10
```

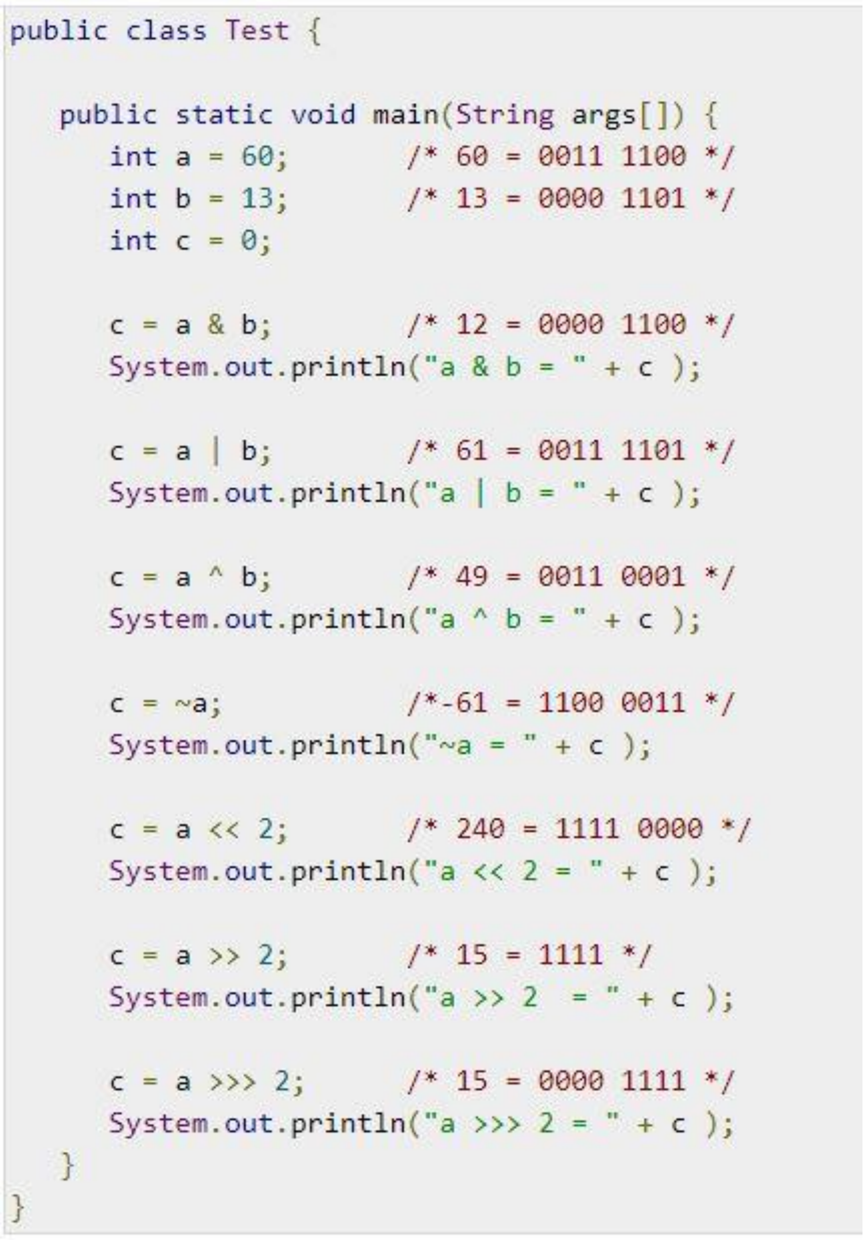
Bitwise Operator:

- Java defines several bitwise operators, which can be applied to the **integer types, long, int, short, char,** and **byte**.
- Bitwise operator works on bits and performs bit-by-bit operation.
- Assume if **a = 60** and **b = 13**; now in binary format they will be as follows−
- a = 0011 1100
- b = 0000 1101

- a&b = 0000 1100
- a|b = 0011 1101
- ~a = 1100 0011

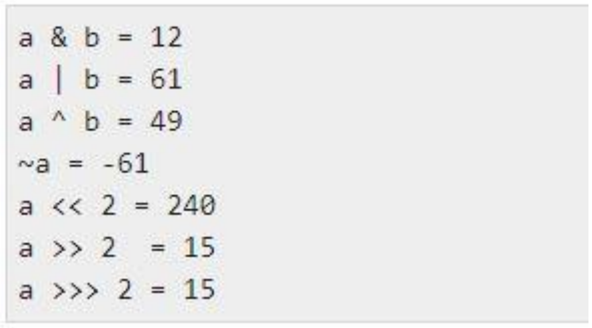
Operator	**Description**	**Example**
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.

Example:



```java
public class Test {

   public static void main(String args[]) {
      int a = 60;        /* 60 = 0011 1100 */
      int b = 13;        /* 13 = 0000 1101 */
      int c = 0;

      c = a & b;         /* 12 = 0000 1100 */
      System.out.println("a & b = " + c );

      c = a | b;         /* 61 = 0011 1101 */
      System.out.println("a | b = " + c );

      c = a ^ b;         /* 49 = 0011 0001 */
      System.out.println("a ^ b = " + c );

      c = ~a;            /*-61 = 1100 0011 */
      System.out.println("~a = " + c );

      c = a << 2;        /* 240 = 1111 0000 */
      System.out.println("a << 2 = " + c );

      c = a >> 2;        /* 15 = 1111 */
      System.out.println("a >> 2  = " + c );

      c = a >>> 2;       /* 15 = 0000 1111 */
      System.out.println("a >>> 2 = " + c );
   }
}
```

Output:



```
a & b = 12
a | b = 61
a ^ b = 49
~a = -61
a << 2 = 240
a >> 2  = 15
a >>> 2 = 15
```

Operator Precedence:

- **Operator precedence** determines the grouping of terms in an **expression**.
- This affects how an expression is evaluated.
- Certain operators have **higher precedence** than **others**; for example, the **multiplication operator (*)** has **higher precedence** than the **addition operator (+)** –
- For example, **x = 7 + 3 * 2**; here x is assigned 13, not 20 because **operator *** has **higher precedence** than **+**, so it first gets multiplied with 3 * 2 and then adds into 7.
- Here, **operators** with the **highest precedence** appear at the **top of the table**, those with the **lowest appear at the bottom**.
- Within an expression, **higher precedence operators** will be **evaluated first**.

Category	Operator	Associativity
Postfix	expression++ expression--	Left to right
Unary	++expression –-expression +expression –expression ~ !	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >> >>>	Left to right
Relational	< > <= >= instanceof	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right

Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= ^= |= <<= >>= >>>=	Right to left

JAVA

UNIT 3: OPERATORS AND CONTROL STATEMENTS

Control Flow in Java:

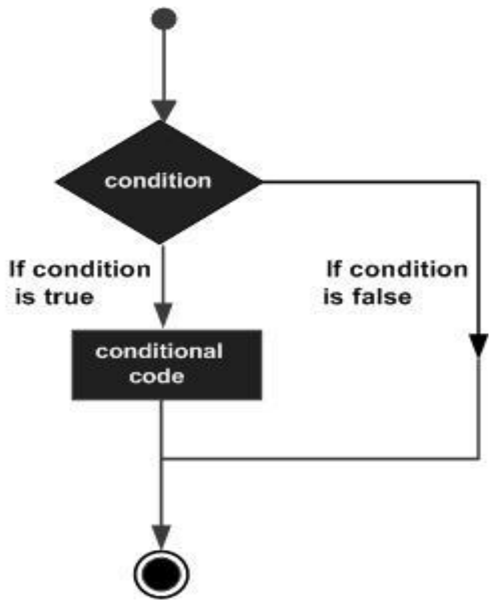
- Java compiler executes the java code from top to bottom.
- The statements are executed according to the order in which they appear.
- Java provides statements that can be used to control the flow of java code. Such statements are called control flow statements.

Java provides **three types** of control flow statements.

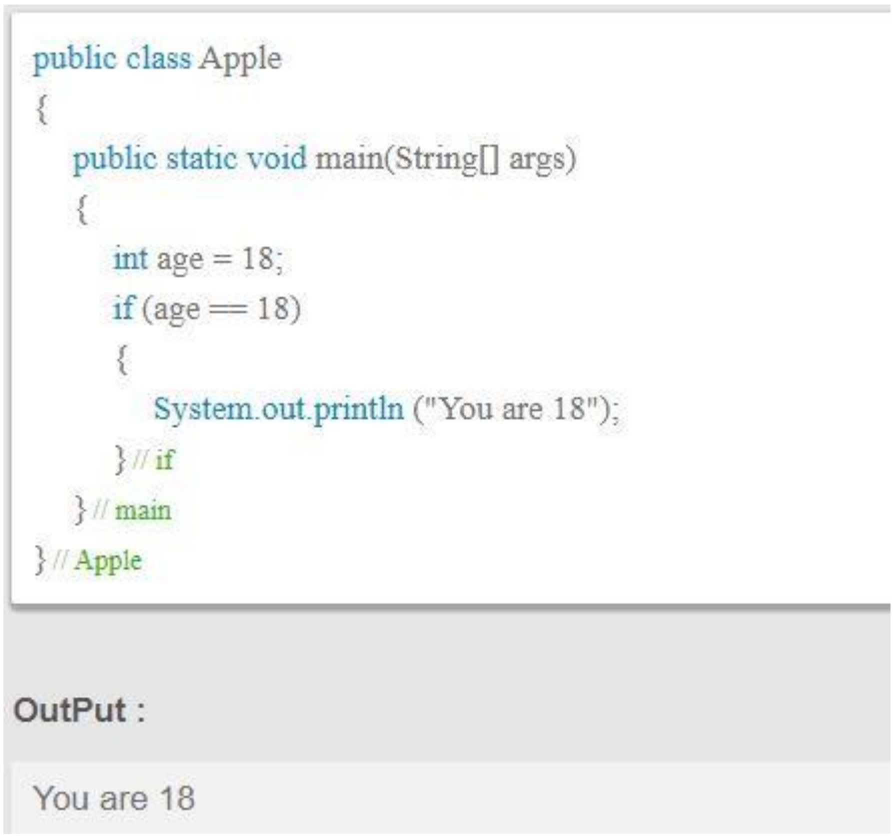
1. **Decision Making statements**
2. **Loop statements**
3. **Jump statements**

1. Decision Making Statements:

- Decision making structures have **one or more conditions** to be **evaluated** or **tested by the program**.
- **statement** that are to be **executed**, **if the condition** is determined to be **true**, or **other statements** to be **executed if the condition** is determined to be **false**.
- There are **two types** of **decision-making statements** in java, I.e., **If statement** and **switch statement**.



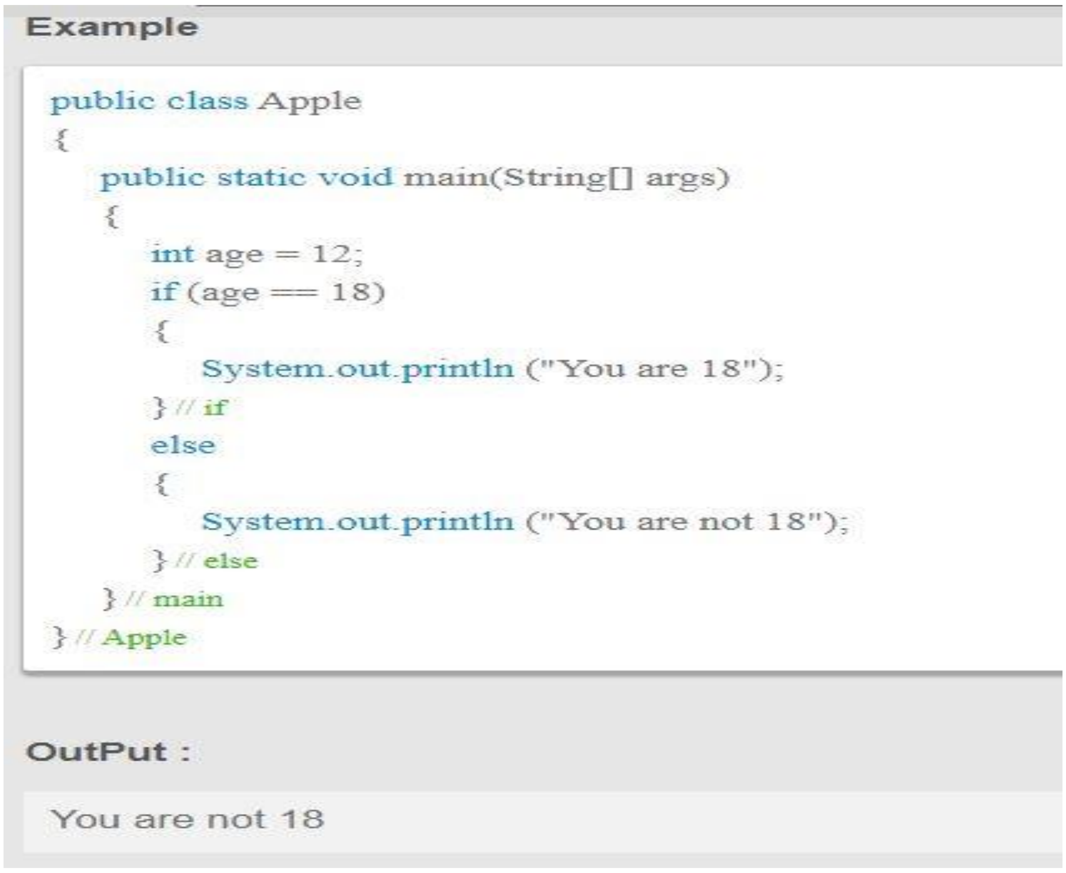
Example:



```
public class Apple
{
    public static void main(String[] args)
    {
        int age = 18;
        if (age == 18)
        {
            System.out.println ("You are 18");
        } // if
    } // main
} // Apple
```

OutPut :

```
You are 18
```



Example

```
public class Apple
{
    public static void main(String[] args)
    {
        int age = 12;
        if (age == 18)
        {
            System.out.println ("You are 18");
        } // if
        else
        {
            System.out.println ("You are not 18");
        } // else
    } // main
} // Apple
```

OutPut :

```
You are not 18
```

If Statement:

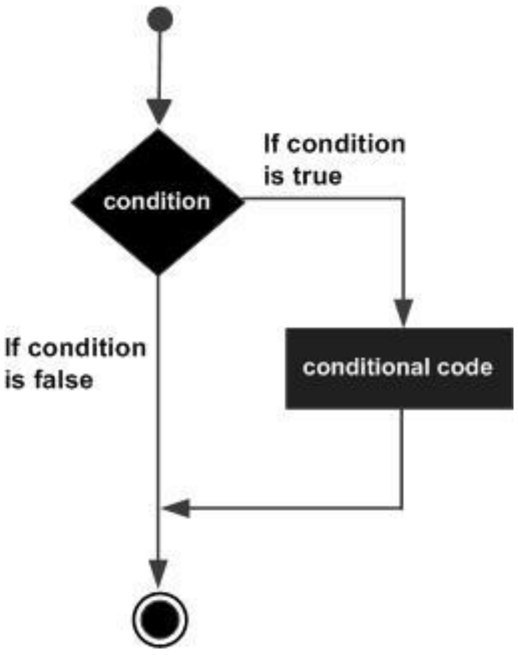
- The **if statement** is used to check some **given condition** and perform some operations depending upon the correctness of that **condition.**
- The Java if statement tests the condition. It executes the ***if block*** if condition is true.

Syntax:

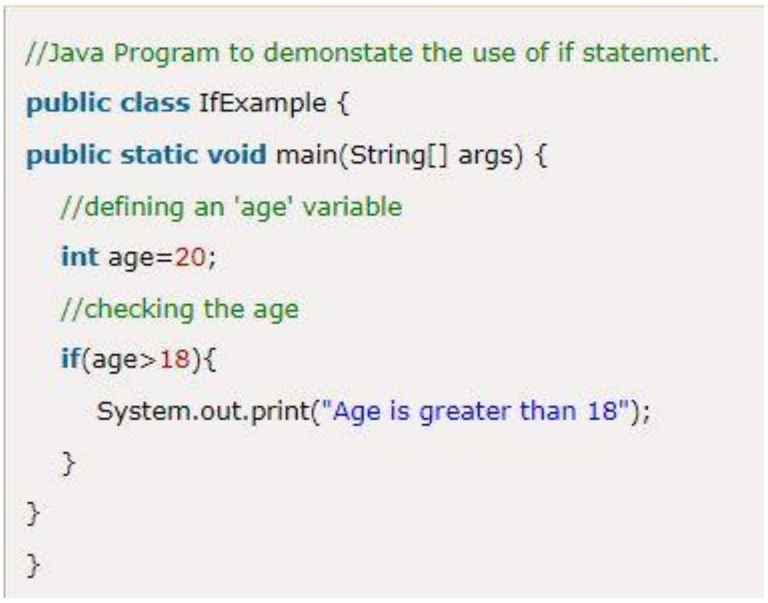
```
if(expression)
{
    statement inside;
}
    statement outside;
```

If the *expression* returns true, then the **statement-inside** will be executed, otherwise **statement-inside** is skipped and only the **statement-outside** is executed.

Flowchart:



Example 1: Program

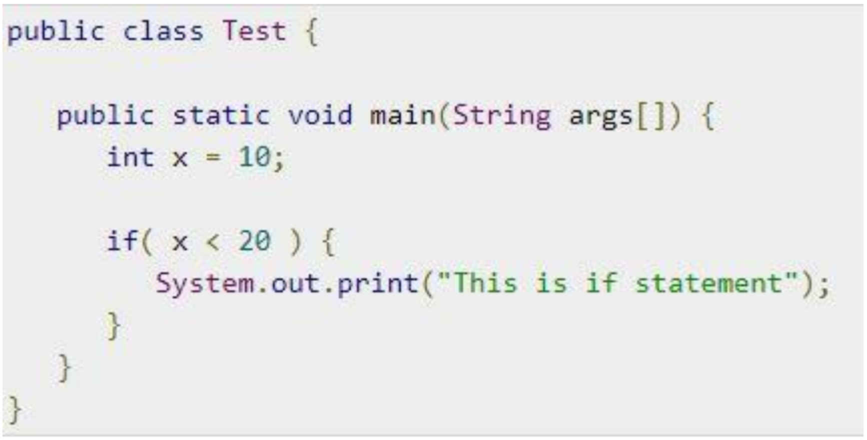


```
//Java Program to demonstate the use of if statement.
public class IfExample {
public static void main(String[] args) {
   //defining an 'age' variable
   int age=20;
   //checking the age
   if(age>18){
      System.out.print("Age is greater than 18");
   }
}
}
```

Output:

```
Age is greater than 18
```

Example 2: Program



```
public class Test {

   public static void main(String args[]) {
      int x = 10;

      if( x < 20 ) {
         System.out.print("This is if statement");
      }
   }
}
```

Output:

```
This is if statement
```

Example 3: Program

```
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y > 20) {
System.out.println("x + y is greater than                20");
}
}


}
```

Output:

```
x + y is greater than 20
```

If-else Statement:

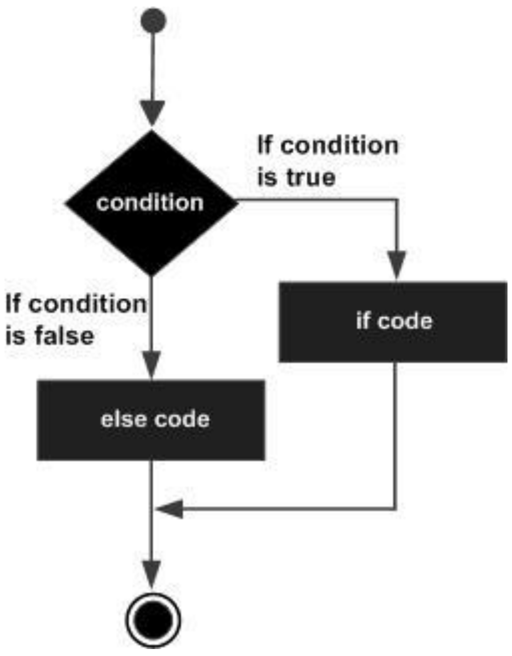
- The **if-else statement** is used to perform two operations for a single condition.
- The **if-else statement** is an extension to the **if statement** using which, we can perform two different operations, i.e., one is for the correctness of that condition, and the other is for the incorrectness of the condition.
- we must notice that if and else block cannot be executed simultaneously.

Syntax:

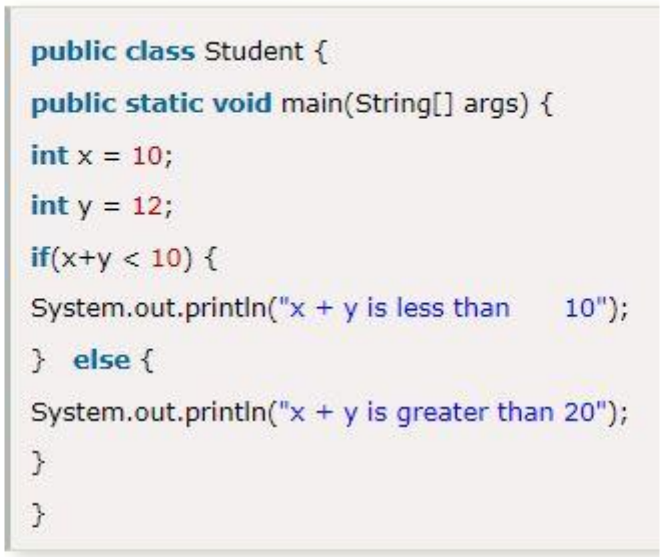
```
if(expression)
{
    statement block1;
}
else
{
    statement block2;
}
```

If the *expression* is true, the **statement-block1** is executed, else **statement-block1** is skipped and **statement-block2** is executed.

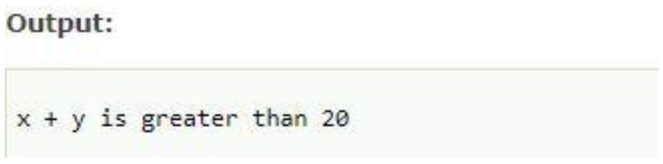
Flowchart:



Example 1: Program



```
public class Student {
public static void main(String[] args) {
int x = 10;
int y = 12;
if(x+y < 10) {
System.out.println("x + y is less than      10");
}   else {
System.out.println("x + y is greater than 20");
}
}
```



Output:

```
x + y is greater than 20
```

Example 2: Program

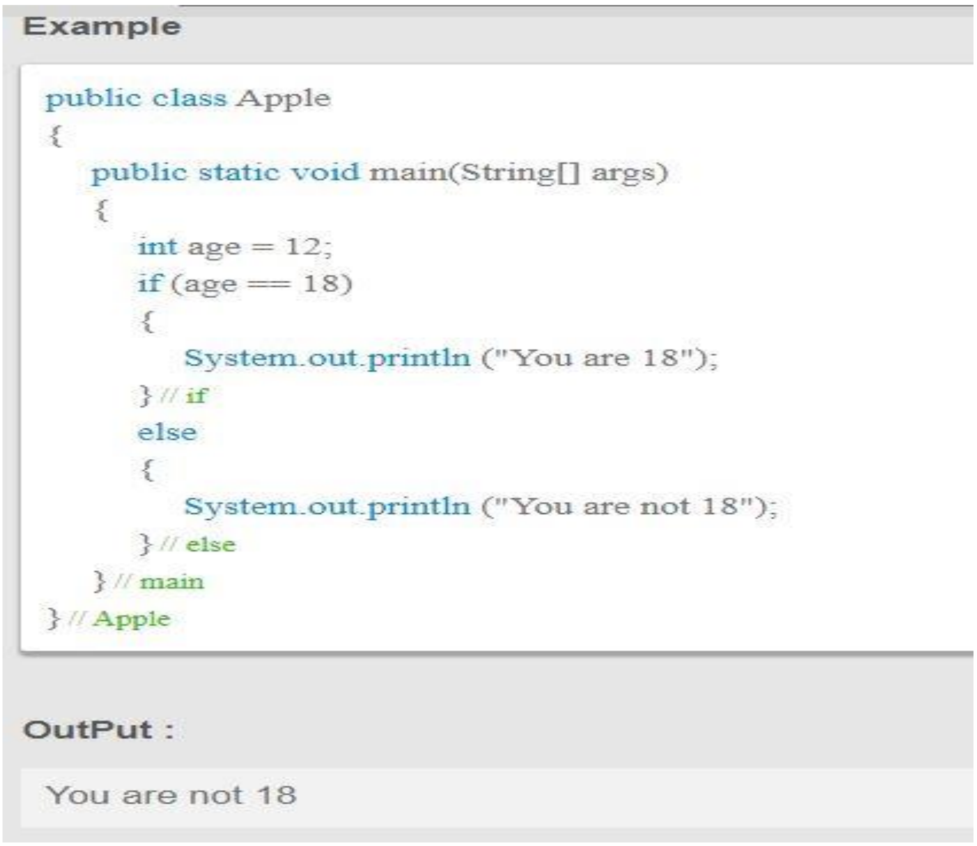
```
public class Test {

   public static void main(String args[]) {
      int x = 30;

      if( x < 20 ) {
         System.out.print("This is if statement");
      }else {
         System.out.print("This is else statement");
      }
   }
}
```

This will produce the following result –

Output

```
This is else statement
```

Example 3: Program



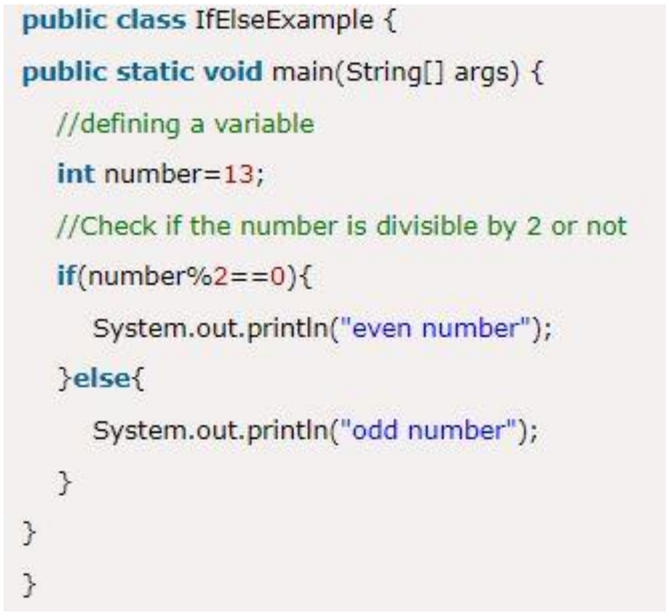
Example

```
public class Apple
{
   public static void main(String[] args)
   {
      int age = 12;
      if (age == 18)
      {
         System.out.println ("You are 18");
      } // if
      else
      {
         System.out.println ("You are not 18");
      } // else
   } // main
} // Apple
```

OutPut :

```
You are not 18
```

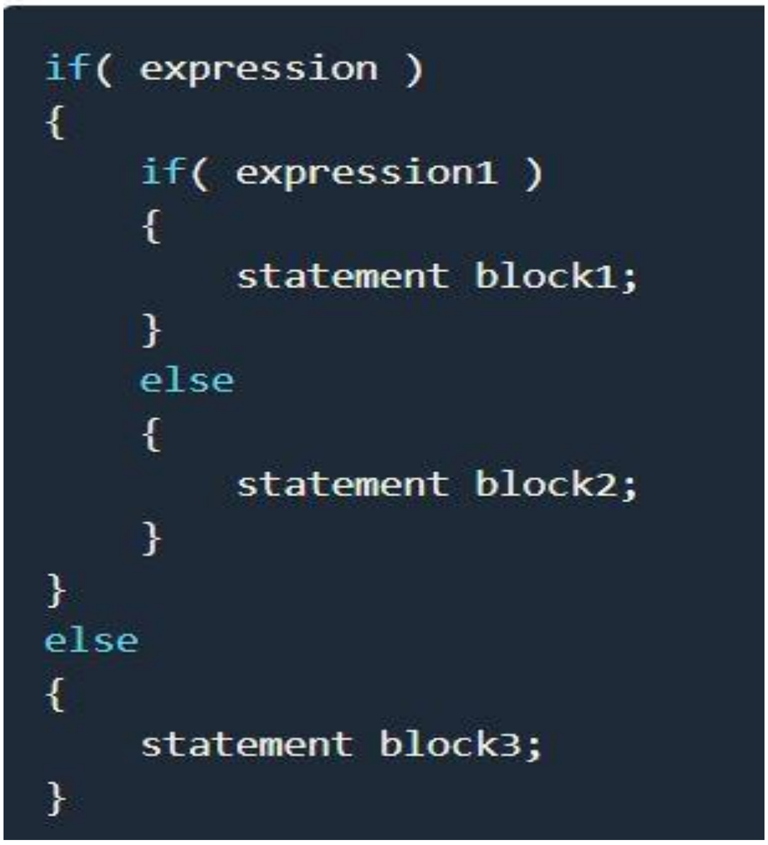
Example 4: Program



```
public class IfElseExample {
public static void main(String[] args) {
    //defining a variable
    int number=13;
    //Check if the number is divisible by 2 or not
    if(number%2==0){
        System.out.println("even number");
    }else{
        System.out.println("odd number");
    }
}
}
```

Nested if:

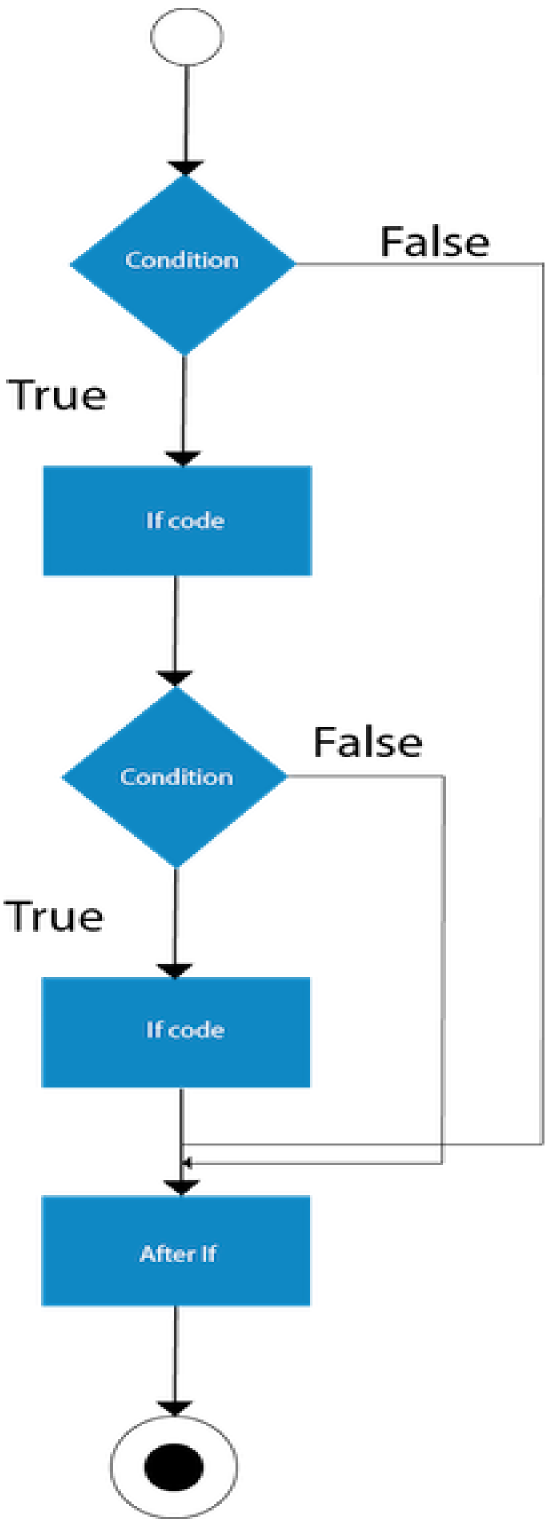
Syntax:



```
if( expression )
{
     if( expression1 )
     {
          statement block1;
     }
     else
     {
          statement block2;
     }
}
else
{
     statement block3;
}
```

if *expression* is false then **statement-block3** will be executed, otherwise the execution continues and enters inside the first `if` to perform the check for the next if block, where if *expression 1* is true the **statement-block1** is executed otherwise **statement-block2** is executed.

Flow chart:



Example 1:

```
//Java Program to demonstrate the use of Nested If Statement.
public class JavaNestedIfExample {
public static void main(String[] args) {
    //Creating two variables for age and weight
    int age=20;
    int weight=80;
    //applying condition on age and weight
    if(age>=18){
        if(weight>50){
            System.out.println("You are eligible to donate blood");
        }
    }
}}
```

Output:

```
You are eligible to donate blood
```

Example 2:

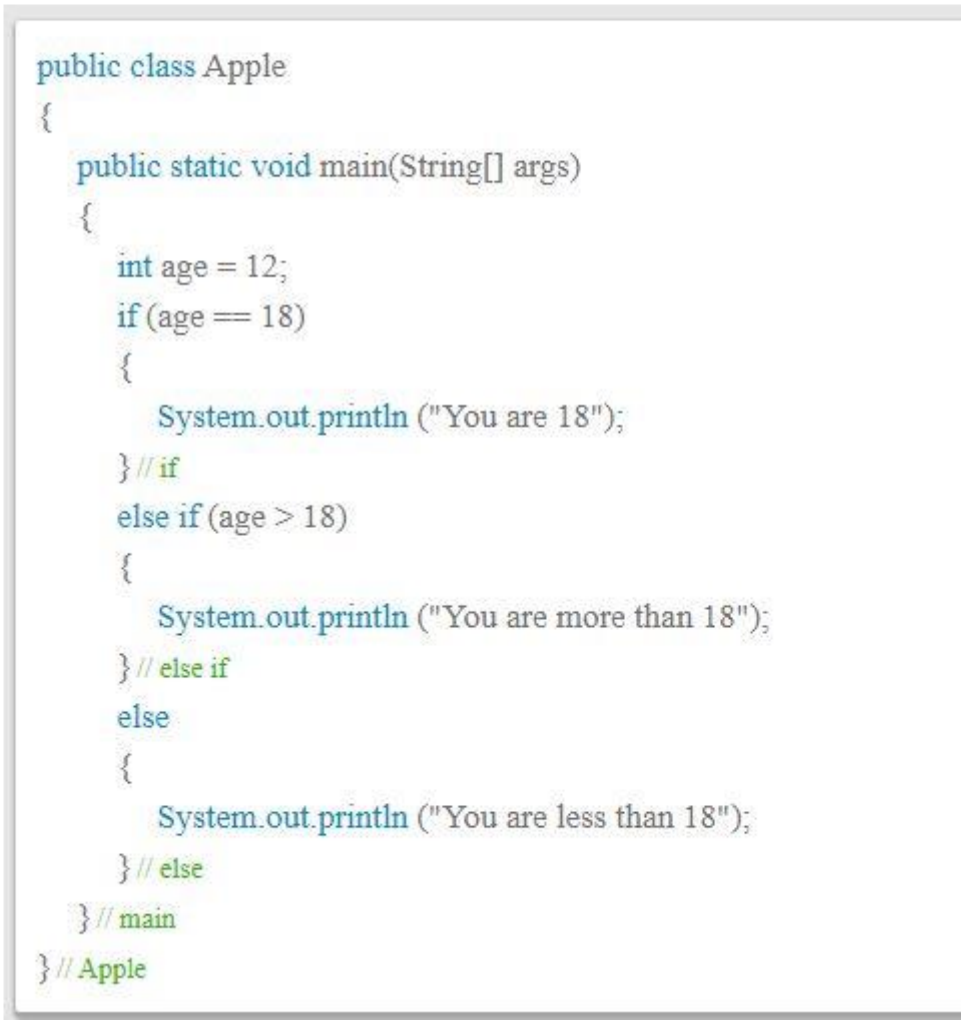
```
public class Test {

   public static void main(String args[]) {
      int x = 30;
      int y = 10;

      if( x == 30 ) {
         if( y == 10 ) {
            System.out.print("X = 30 and Y = 10");
         }
      }
   }
}
```

This will produce the following result –

Output

```
X = 30 and Y = 10
```

Example 3:



```
public class Apple
{
    public static void main(String[] args)
    {
        int age = 12;
        if (age == 18)
        {
            System.out.println ("You are 18");
        } // if
        else if (age > 18)
        {
            System.out.println ("You are more than 18");
        } // else if
        else
        {
            System.out.println ("You are less than 18");
        } // else
    } // main
} // Apple
```

Output:

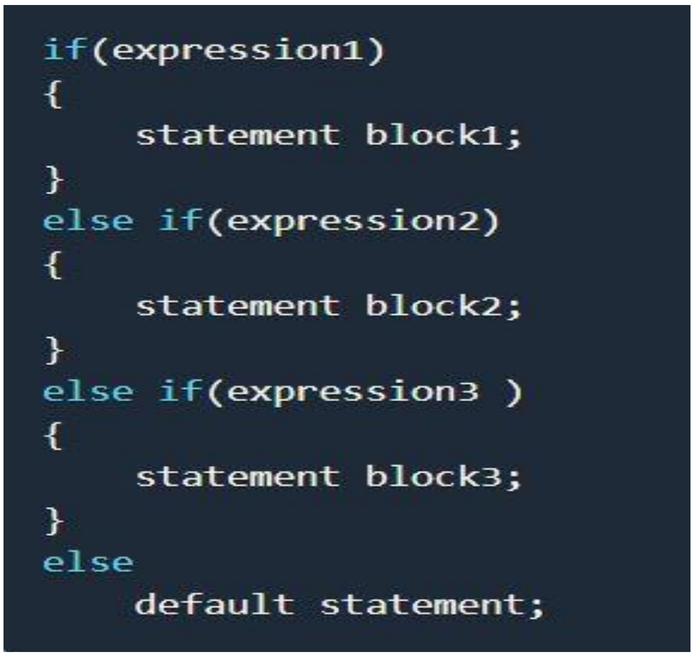
```
You are less than 18
```

If else if ladder:

- The if-else-if ladder statement is an extension to the if-else statement.
- It is used in the scenario where there are multiple cases to be performed for different conditions.
- In if-else-if ladder statement, if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed.
- There are multiple else-if blocks possible.

- It is similar to the switch case statement where the default is executed instead of else block if none of the cases is matched.

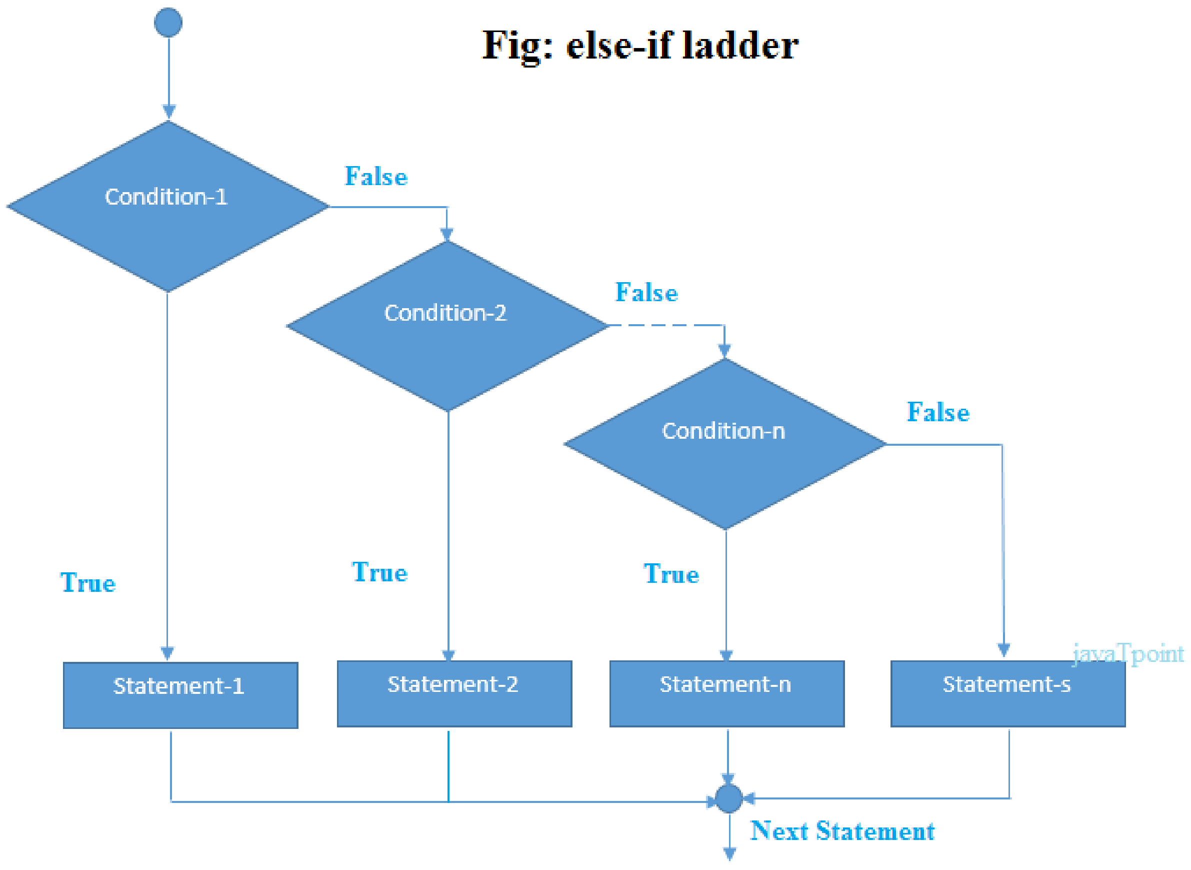
Syntax:



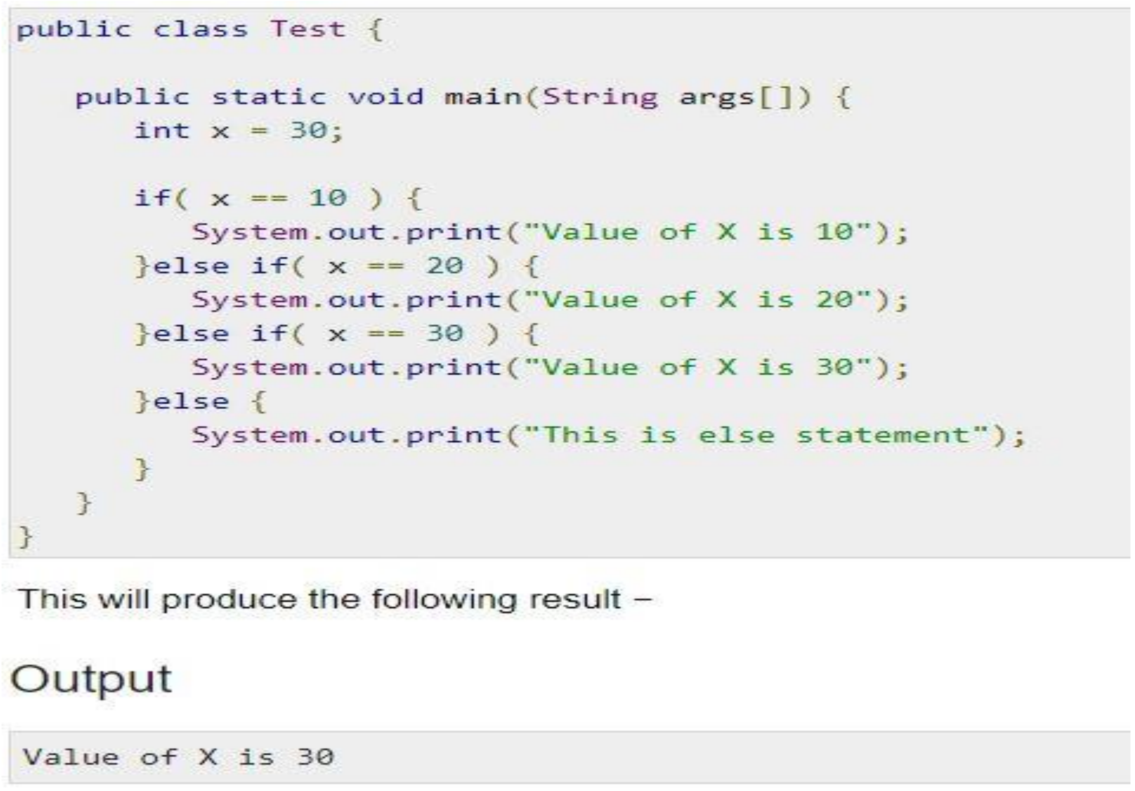
```
if(expression1)
{
        statement block1;
}
else if(expression2)
{
        statement block2;
}
else if(expression3 )
{
        statement block3;
}
else
        default statement;
```

Flowchart:

Fig: else-if ladder



Example 1:



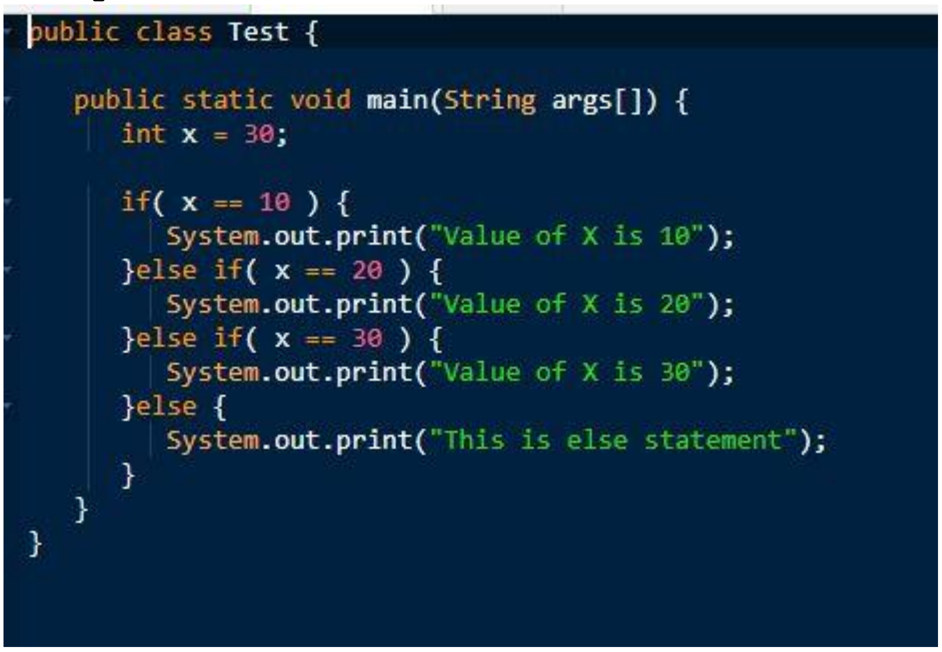
```
public class Test {

   public static void main(String args[]) {
      int x = 30;

      if( x == 10 ) {
         System.out.print("Value of X is 10");
      }else if( x == 20 ) {
         System.out.print("Value of X is 20");
      }else if( x == 30 ) {
         System.out.print("Value of X is 30");
      }else {
         System.out.print("This is else statement");
      }
   }
}
```

This will produce the following result –

Output

```
Value of X is 30
```

Example 2:

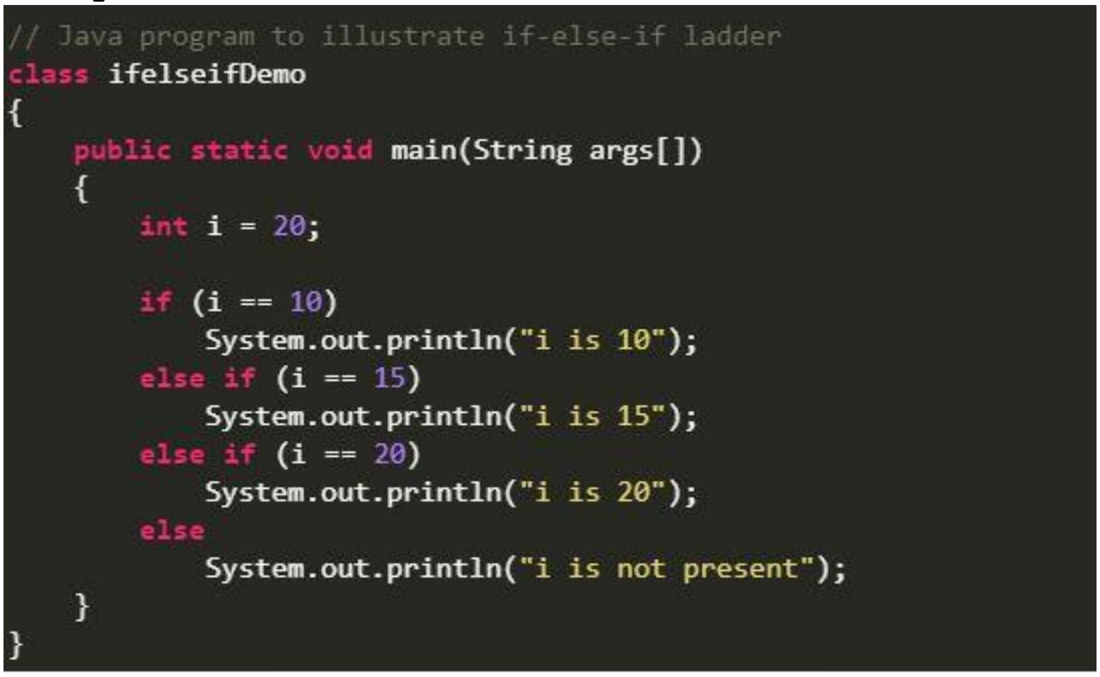


```
public class Test {

   public static void main(String args[]) {
      int x = 30;

      if( x == 10 ) {
         System.out.print("Value of X is 10");
      }else if( x == 20 ) {
         System.out.print("Value of X is 20");
      }else if( x == 30 ) {
         System.out.print("Value of X is 30");
      }else {
         System.out.print("This is else statement");
      }
   }
}
```

Output

```
Value of X is 30
```

Example 3:

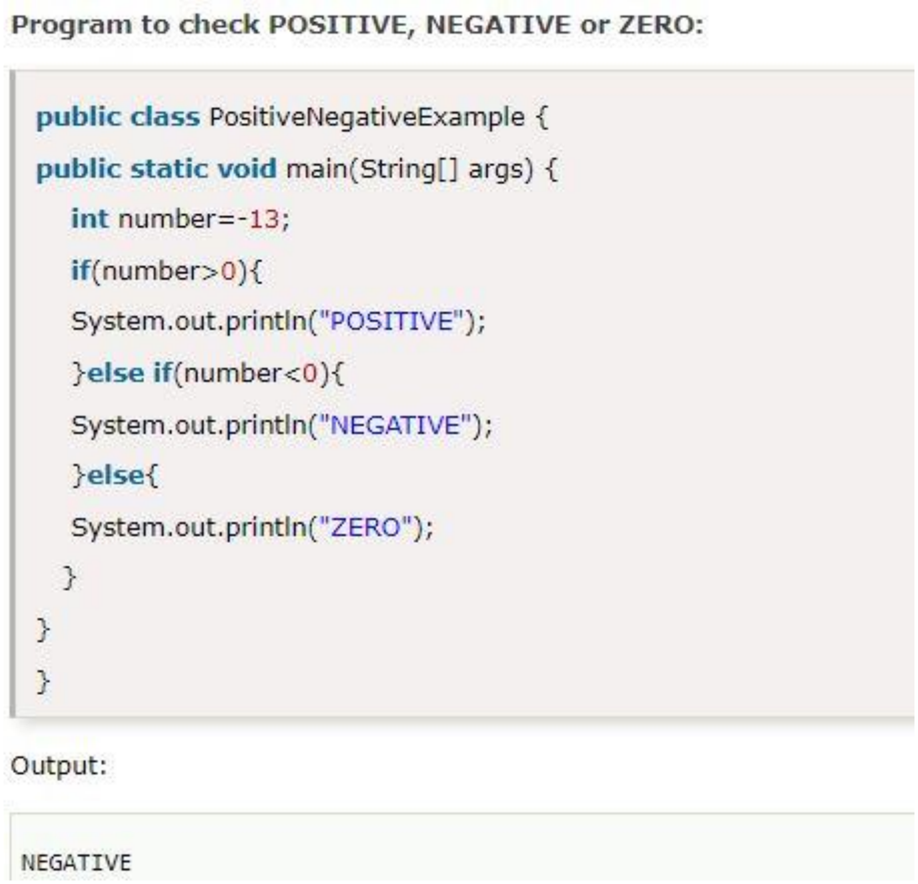


```java
// Java program to illustrate if-else-if ladder
class ifelseifDemo
{
    public static void main(String args[])
    {
        int i = 20;

        if (i == 10)
            System.out.println("i is 10");
        else if (i == 15)
            System.out.println("i is 15");
        else if (i == 20)
            System.out.println("i is 20");
        else
            System.out.println("i is not present");
    }
}
```

Output:

```
i is 20
```

Example 4:



Program to check POSITIVE, NEGATIVE or ZERO:

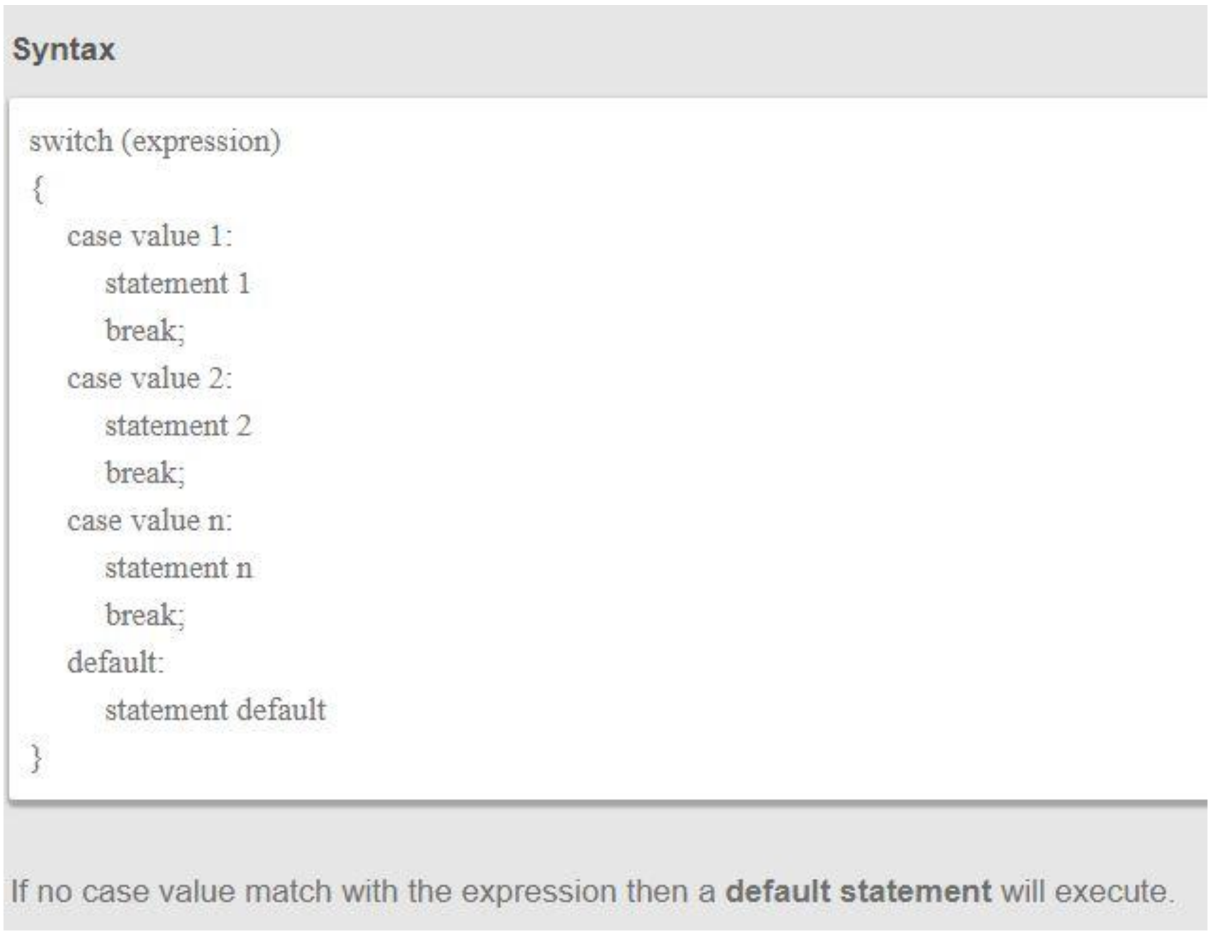
```java
public class PositiveNegativeExample {
public static void main(String[] args) {
    int number=-13;
    if(number>0){
    System.out.println("POSITIVE");
    }else if(number<0){
    System.out.println("NEGATIVE");
    }else{
    System.out.println("ZERO");
   }
}
}
```

Output:

```
NEGATIVE
```

Switch Statements:

- The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch statement works with byte, short, int, long, enum types, String.
- A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

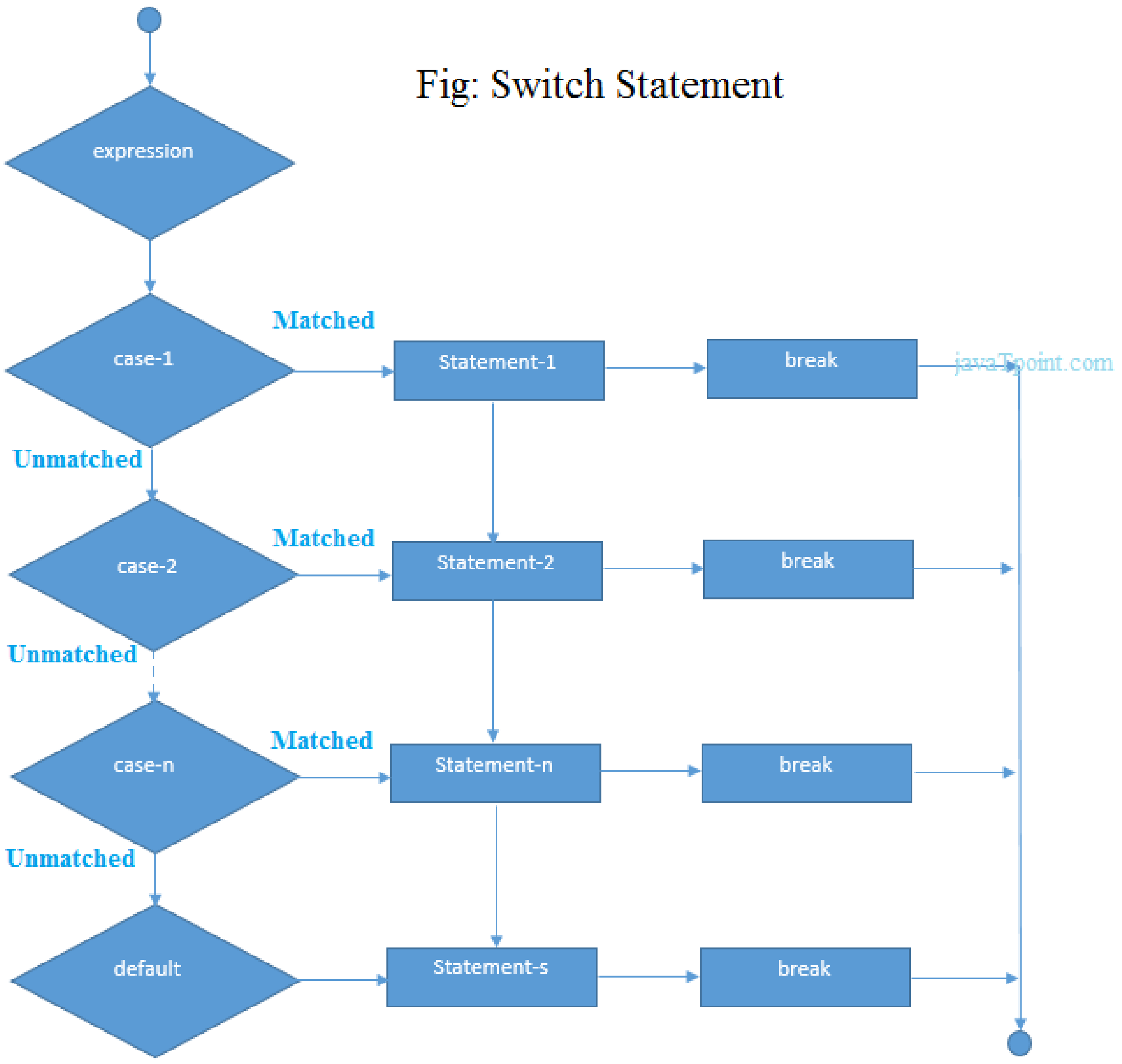


Syntax

```
switch (expression)
{
    case value 1:
        statement 1
        break;
    case value 2:
        statement 2
        break;
    case value n:
        statement n
        break;
    default:
        statement default
}
```

If no case value match with the expression then a **default statement** will execute.

Flow Chart:



Rules for switch statement in C language

1) The *switch expression* must be of an integer or character type.

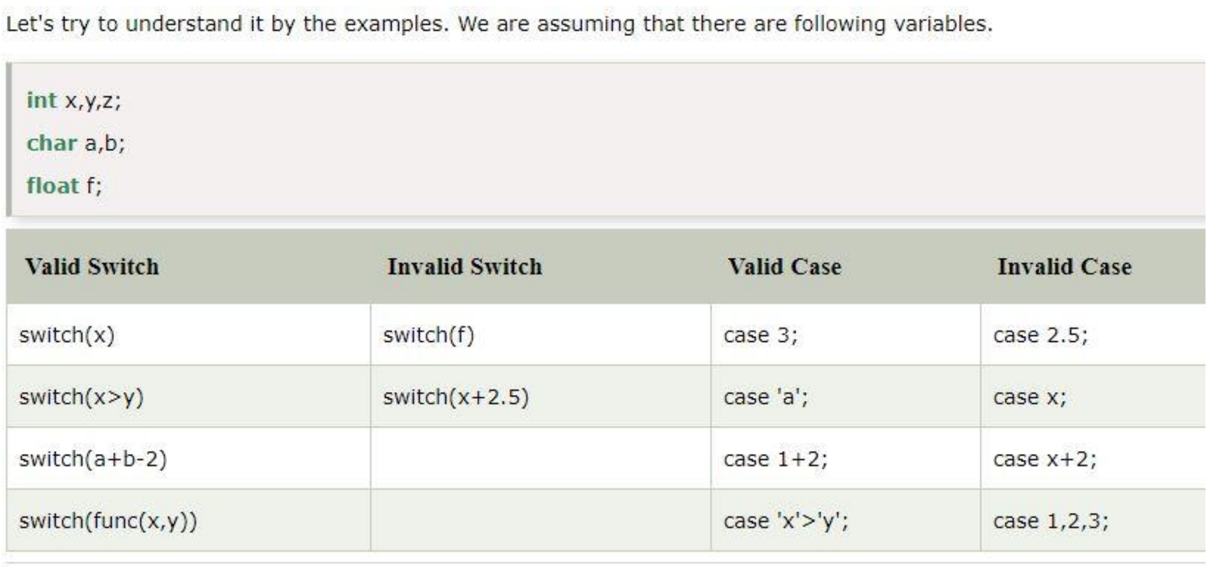
2) The *case value* must be an integer or character constant.

3) The *case value* can be used only inside the switch statement.

4) The *break statement* in switch case is not must.

5) break statement is optional.

6) If there is no break statement found in the case, all the cases will be executed present after the matched case.

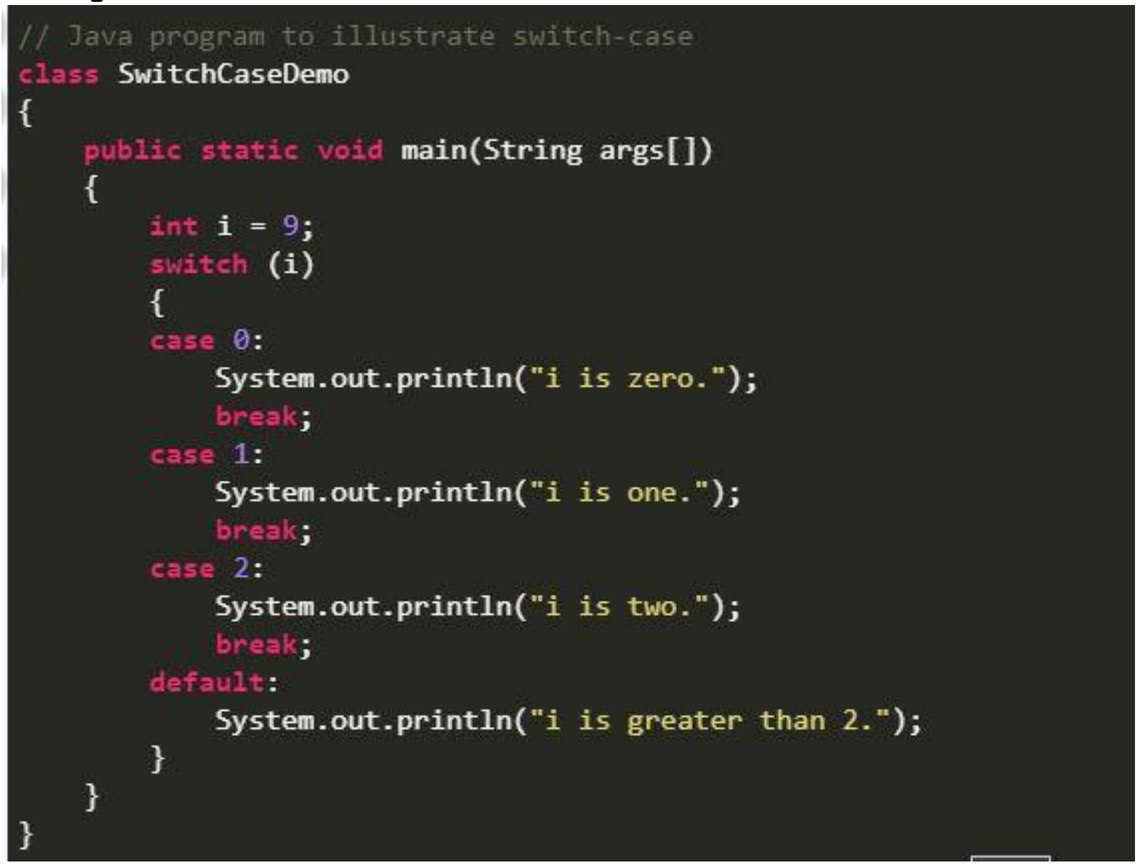


Let's try to understand it by the examples. We are assuming that there are following variables.

```
int x,y,z;
char a,b;
float f;
```

Valid Switch	Invalid Switch	Valid Case	Invalid Case
switch(x)	switch(f)	case 3;	case 2.5;
switch(x>y)	switch(x+2.5)	case 'a';	case x;
switch(a+b-2)		case 1+2;	case x+2;
switch(func(x,y))		case 'x'>'y';	case 1,2,3;

Example 1:

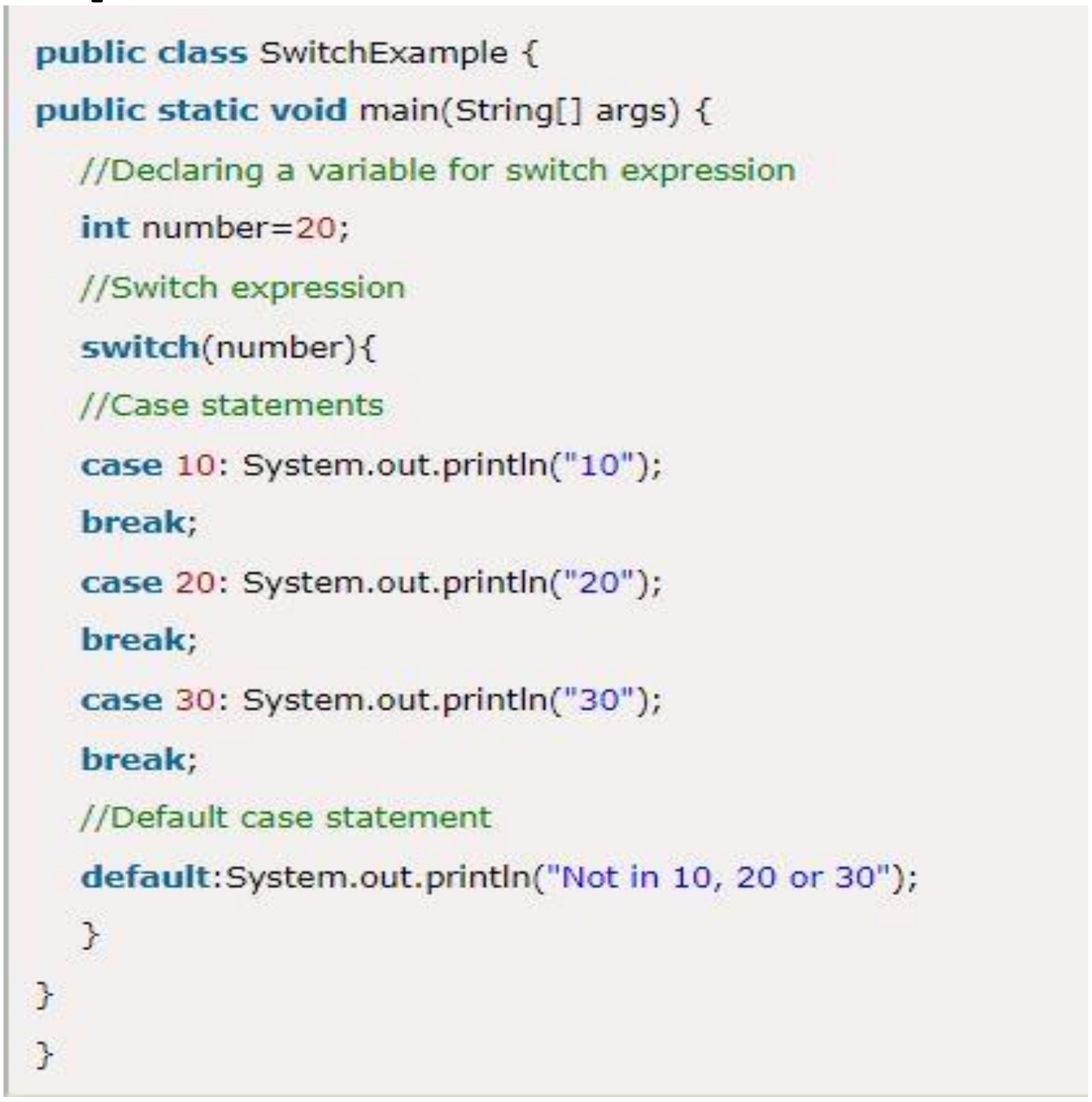


```
// Java program to illustrate switch-case
class SwitchCaseDemo
{
    public static void main(String args[])
    {
        int i = 9;
        switch (i)
        {
        case 0:
            System.out.println("i is zero.");
            break;
        case 1:
            System.out.println("i is one.");
            break;
        case 2:
            System.out.println("i is two.");
            break;
        default:
            System.out.println("i is greater than 2.");
        }
    }
}
```

Output:

```
i is greater than 2.
```

Example 2:

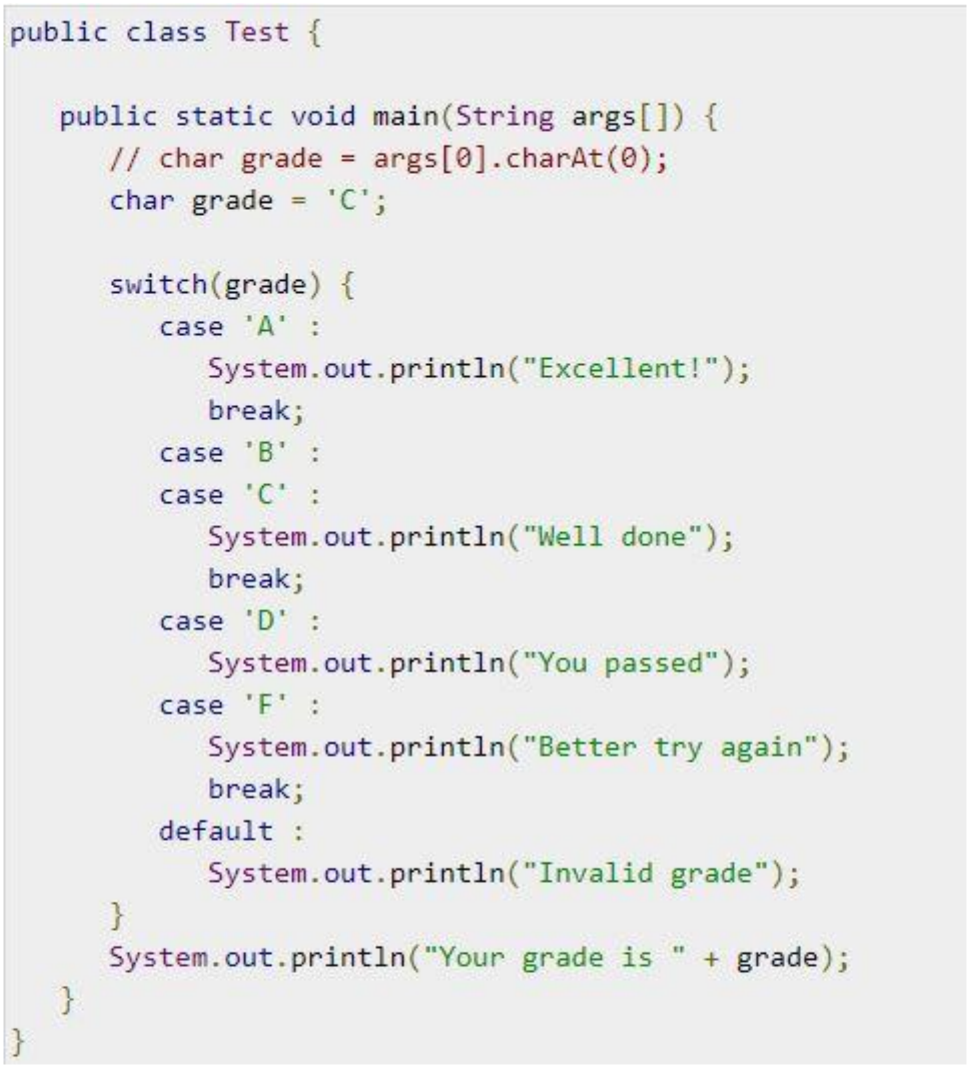


```
public class SwitchExample {
public static void main(String[] args) {
    //Declaring a variable for switch expression
    int number=20;
    //Switch expression
    switch(number){
    //Case statements
    case 10: System.out.println("10");
    break;
    case 20: System.out.println("20");
    break;
    case 30: System.out.println("30");
    break;
    //Default case statement
    default:System.out.println("Not in 10, 20 or 30");
    }
}
}
```

Output:

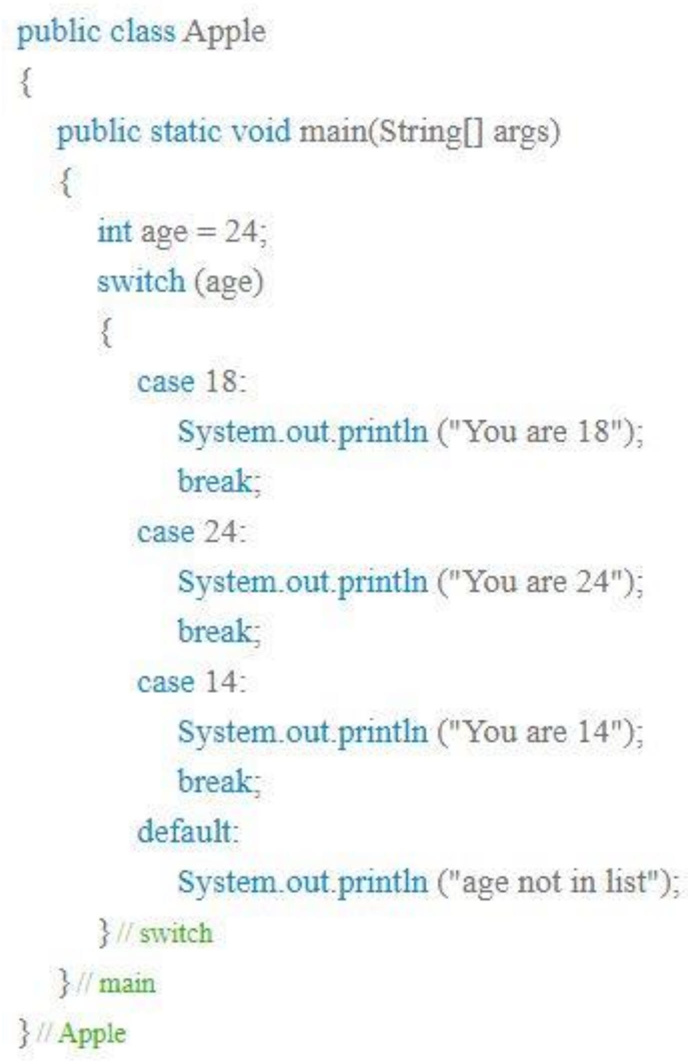
```
20
```

Example 3:



```java
public class Test {

   public static void main(String args[]) {
      // char grade = args[0].charAt(0);
      char grade = 'C';

      switch(grade) {
         case 'A' :
            System.out.println("Excellent!");
            break;
         case 'B' :
         case 'C' :
            System.out.println("Well done");
            break;
         case 'D' :
            System.out.println("You passed");
         case 'F' :
            System.out.println("Better try again");
            break;
         default :
            System.out.println("Invalid grade");
      }
      System.out.println("Your grade is " + grade);
   }
}
```

Example 4:



```
public class Apple
{
    public static void main(String[] args)
    {
        int age = 24;
        switch (age)
        {
            case 18:
                System.out.println ("You are 18");
                break;
            case 24:
                System.out.println ("You are 24");
                break;
            case 14:
                System.out.println ("You are 14");
                break;
            default:
                System.out.println ("age not in list");
        } // switch
    } // main
} // Apple
```

Output:

```
You are 24
```

Example 5:

```
public class SwitchMonthExample {
public static void main(String[] args) {
    int month=8;
    String monthString="";
    switch(month){
    case 1: monthString="1 - January";
    break;
    case 2: monthString="2 - February";
    break;
    case 3: monthString="3 - March";
    break;
    case 4: monthString="4 - April";
    break;
    case 5: monthString="5 - May";
    break;
    case 6: monthString="6 - June";
    break;
    case 7: monthString="7 - July";
    break;
    case 8: monthString="8 - August";
    break;
    case 9: monthString="9 - September";
    break;
    case 10: monthString="10 - October";
    break;
    case 11: monthString="11 - November";
    break;
    case 12: monthString="12 - December";
    break;
    default:System.out.println("Invalid Month");
    }
    System.out.println(monthString);
}
}
```

Output:

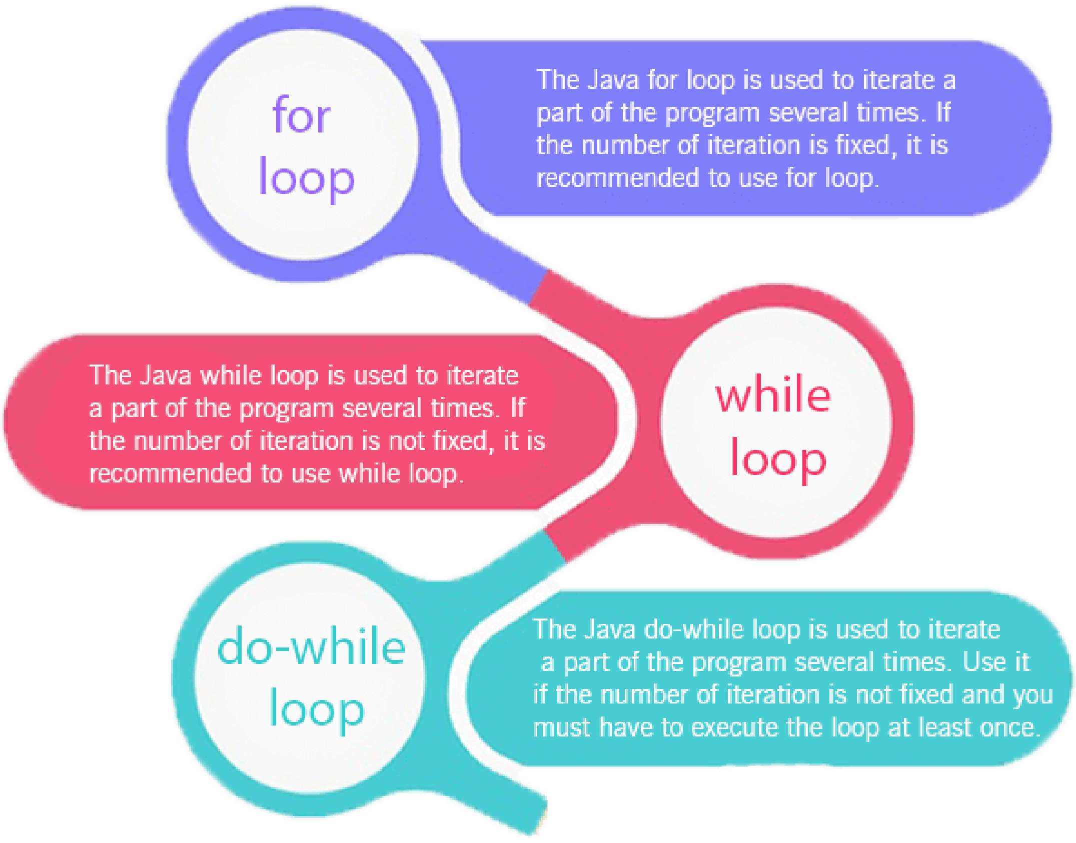
```
8 - August
```

UNIT 3: OPERATORS AND CONTROL STATEMENTS

Iteration/Loop Statements:

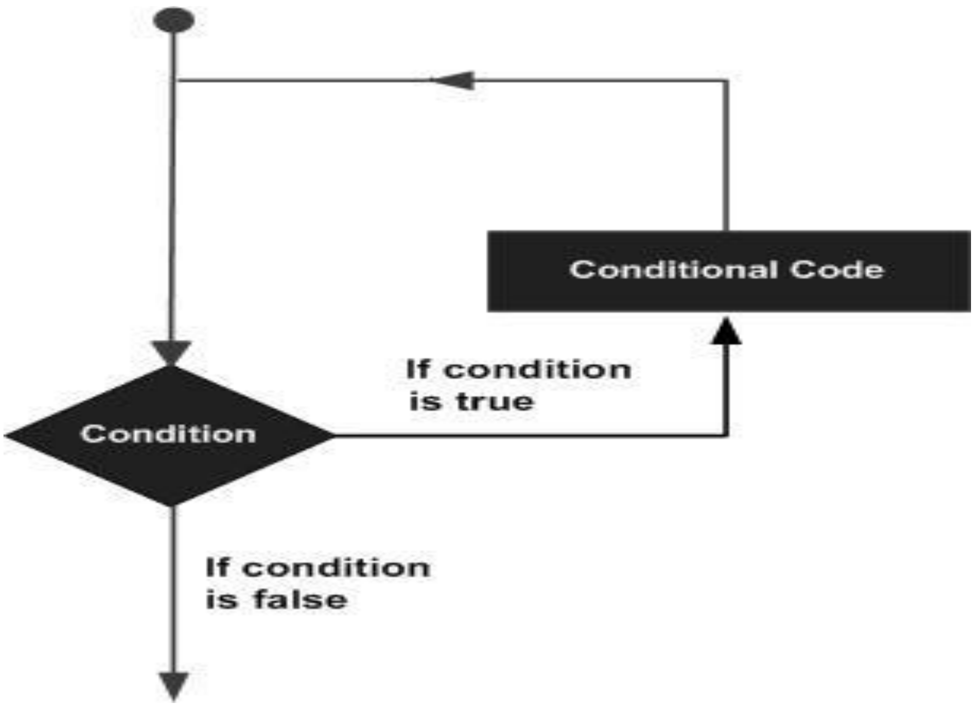
- When you need to execute a block of code several number of times.
- Loops are used to execute a set of instructions/functions repeatedly when some conditions become true.
- In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.
- A **loop** statement allows us to execute a statement or group of statements multiple times.
- There are three types of loops in Java.

1. **for loop**
2. **while loop**
3. **do-while loop**


for loop
The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.
while loop
The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.
do-while loop
The Java do-while loop is used to iterate a part of the program several times. Use it if the number of iteration is not fixed and you must have to execute the loop at least once.

Java For Loop vs While Loop vs Do While Loop:

Comparison	for loop	while loop	do while loop
Introduction	The Java for loop is a control flow statement that iterates a part of the programs multiple times.	The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given Boolean condition.	The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given Boolean condition.
When to use	If the number of iterations is fixed, it is recommended to use for loop.	If the number of iterations is not fixed, it is recommended to use while loop.	If the number of iterations is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.
Syntax	`for(init;condition;inc/dec){ // code to be executed }`	`while(condition){ //code to be executed }`	`do{ //code to be executed }while(condition);`
Example	`//for loop for(int i=1;i<=10;i++){ System.out.println(i); }`	`//while loop int i=1; while(i<=10){ System.out.println(i); i++; }`	`//do-while loop int i=1; do{ System.out.println(i); i++; }while(i<=10);`
Syntax for infinitive loop	`for(;;){ //code to be executed }`	`while(true){ //code to be executed }`	`do{ //code to be executed }while(true);`



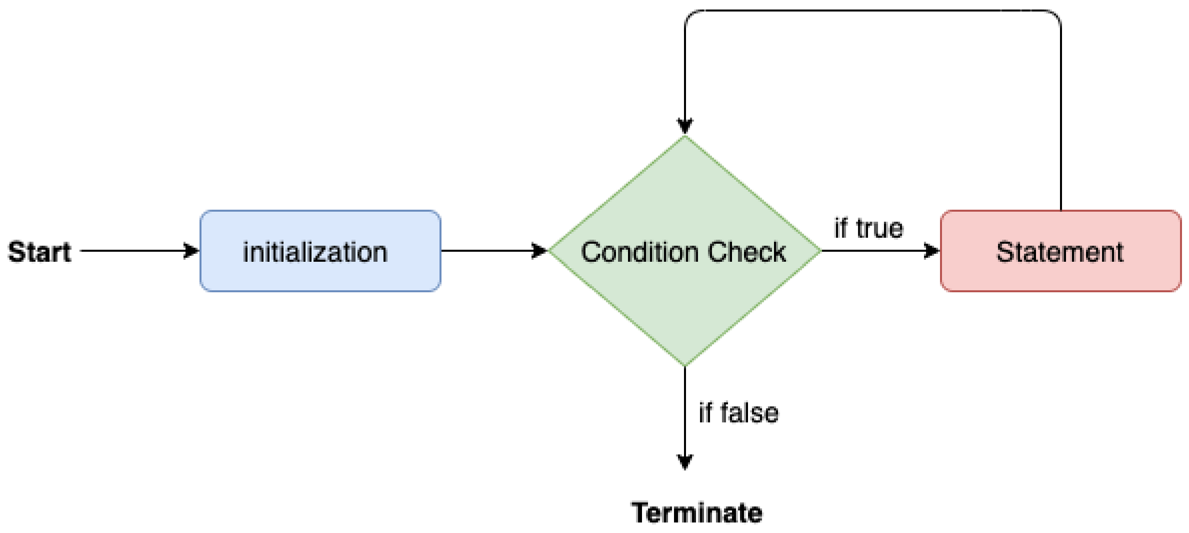
1. For loop:

- A **for loop** is a repetition control structure that allows you to efficiently write a loop that needs to be executed a specific number of times.
- A for loop is useful when you know how many times a task is to be repeated.
- If the number of iteration is fixed, it is recommended to use for loop.
- It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code.
- It consists of four parts:
  - **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable.
  - **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false.
  - **Statement**: The statement of the loop is executed each time until the second condition is false.
  - **Increment/Decrement**: It increments or decrements the variable value.

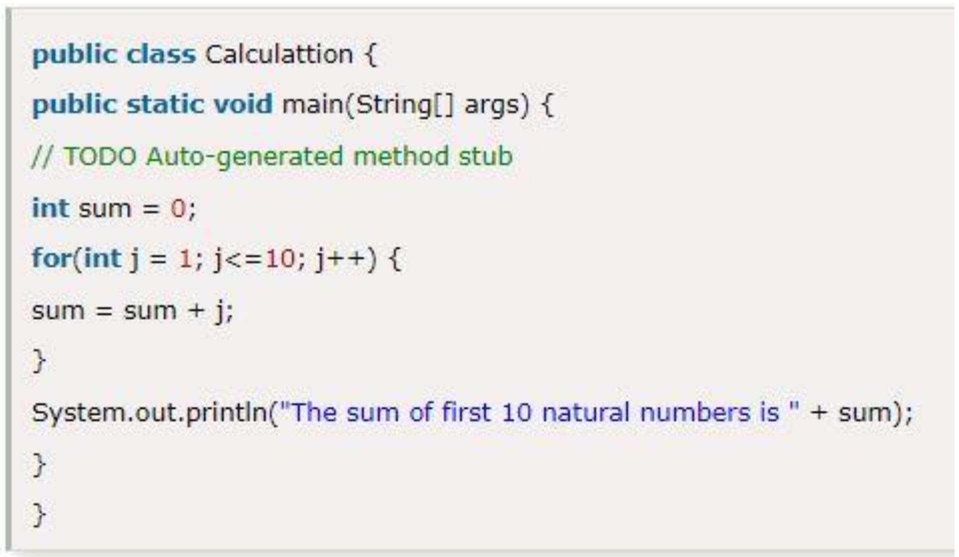
Syntax:

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

Flow chart:



Example 1:



```
public class Calculattion {
public static void main(String[] args) {
// TODO Auto-generated method stub
int sum = 0;
for(int j = 1; j<=10; j++) {
sum = sum + j;
}
System.out.println("The sum of first 10 natural numbers is " + sum);
}
}
```

Output:

```
The sum of first 10 natural numbers is 55
```

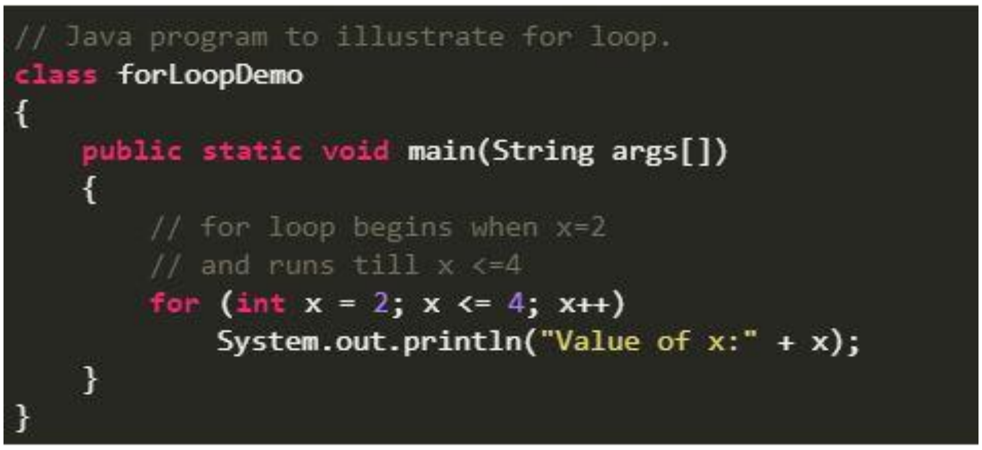
Example 2:

```
public class Test
{

   public static void main(String args[])
   {

      for(int x = 10; x < 20; x = x + 1)
       {
          System.out.print("value of x : " + x );
          System.out.print("\n");
        }
   }
}
```

Output:

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

Example 3:

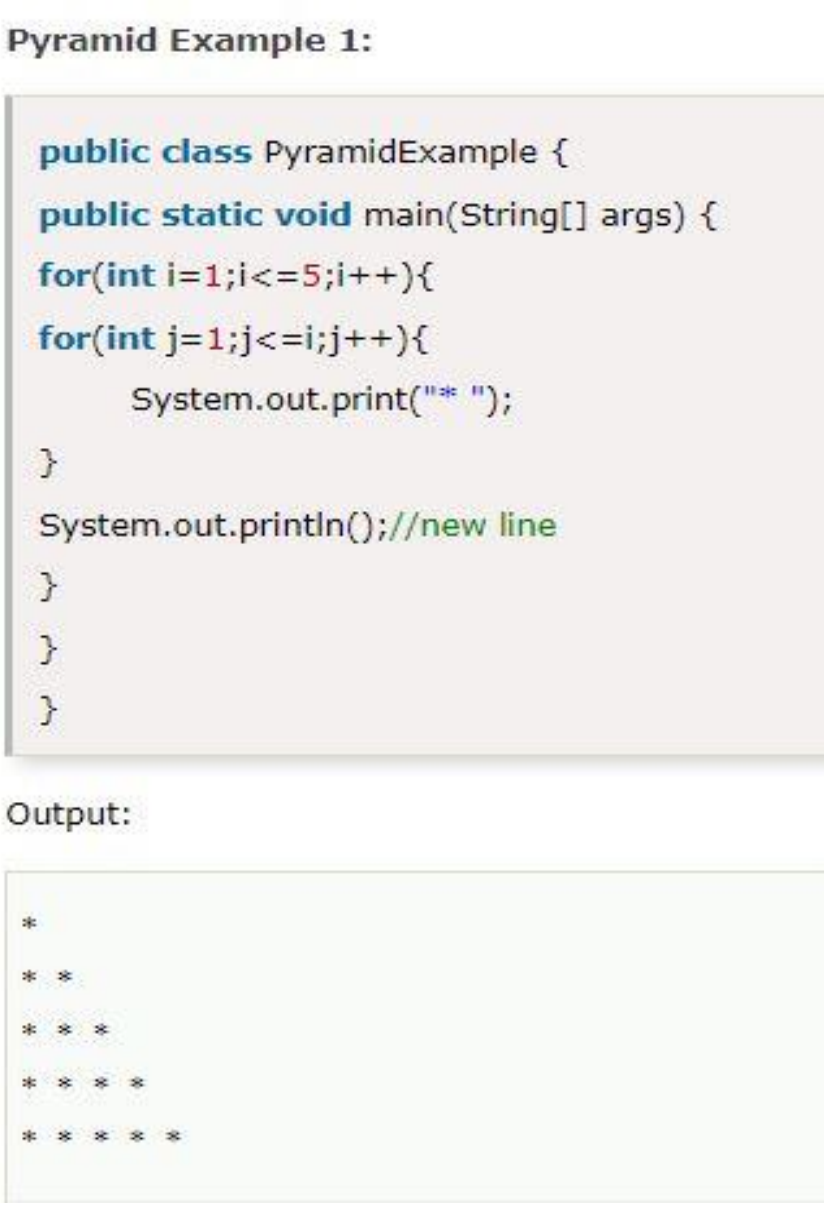


```
// Java program to illustrate for loop.
class forLoopDemo
{
    public static void main(String args[])
    {
        // for loop begins when x=2
        // and runs till x <=4
        for (int x = 2; x <= 4; x++)
            System.out.println("Value of x:" + x);
    }
}
```

Output:

```
Value of x:2

Value of x:3

Value of x:4
```

Example 4:



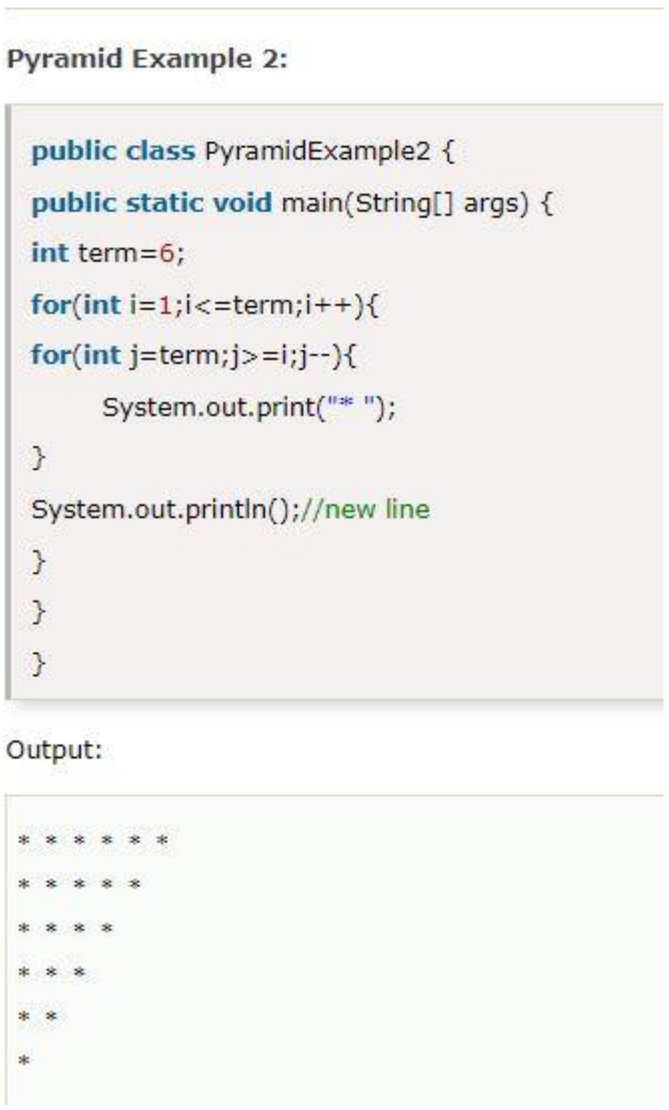
Pyramid Example 1:

```
public class PyramidExample {
public static void main(String[] args) {
for(int i=1;i<=5;i++){
for(int j=1;j<=i;j++){
        System.out.print("* ");
}
System.out.println();//new line
}
}
}
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

Example 5:



Pyramid Example 2:

```
public class PyramidExample2 {
public static void main(String[] args) {
int term=6;
for(int i=1;i<=term;i++){
for(int j=term;j>=i;j--){
        System.out.print("* ");
}
System.out.println();//new line
}
}
}
```

Output:

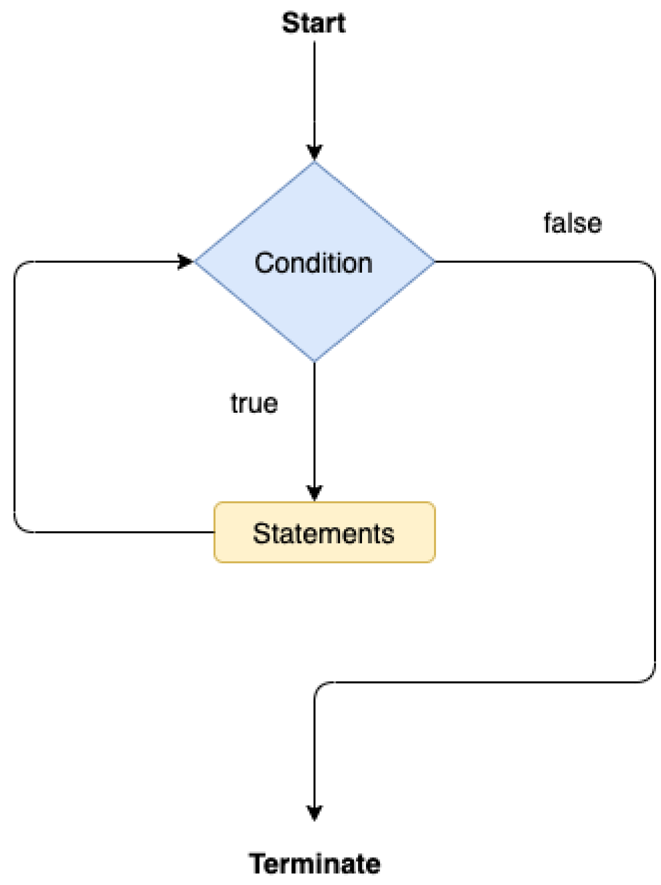
```
* * * * * *
* * * * *
* * * *
* * *
* *
*
```

2. while loop:

- A **while loop** statement in Java programming language repeatedly executes a target statement as long as a given condition is true.
- If the number of iteration is not fixed, it is recommended to use while loop.
- The while loop is also used to iterate over the number of statements multiple times.
- It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

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

Flow Chart:



Example 1:

```
public class WhileExample
{
     public static void main(String[] args)
     {
            int i=1;
            while(i<=10){
```

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

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Example 2:

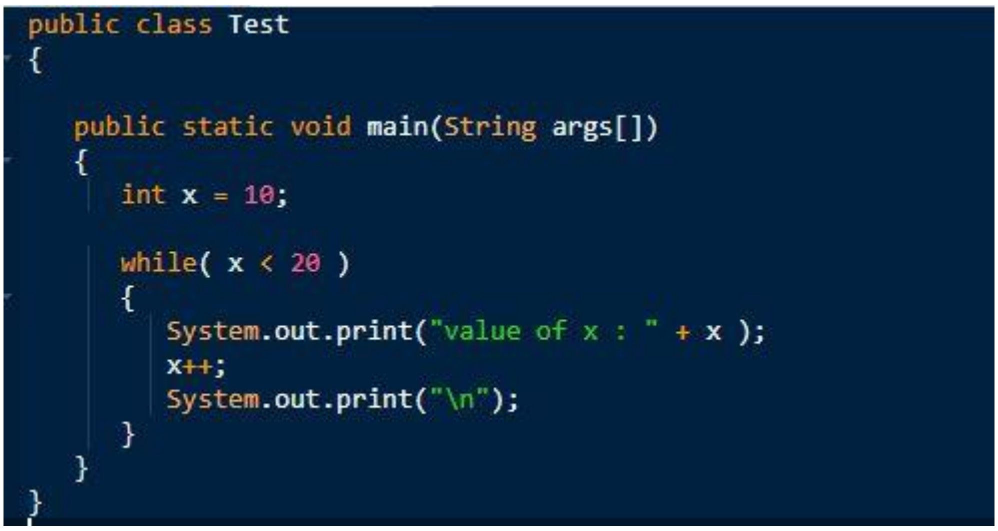
```
public class Calculattion
{
        public static void main(String[] args)
        {
               int i = 0;
               System.out.println("Printing the list of first 10 even numbers \n");
              while(i<=10)
              {
                  System.out.println(i);
                   i = i + 2;
               }
         }
}
```

Output:

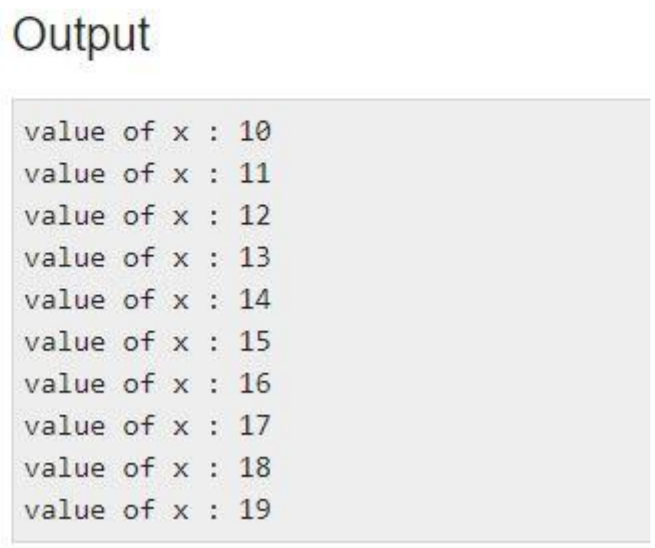
```
Printing the list of first 10 even numbers

0
2
4
```

```
6
8
10
```

Example 3:



```
public class Test
{

   public static void main(String args[])
   {
      int x = 10;

      while( x < 20 )
      {
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }
   }
}
```



Output

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

3. Do while loop:

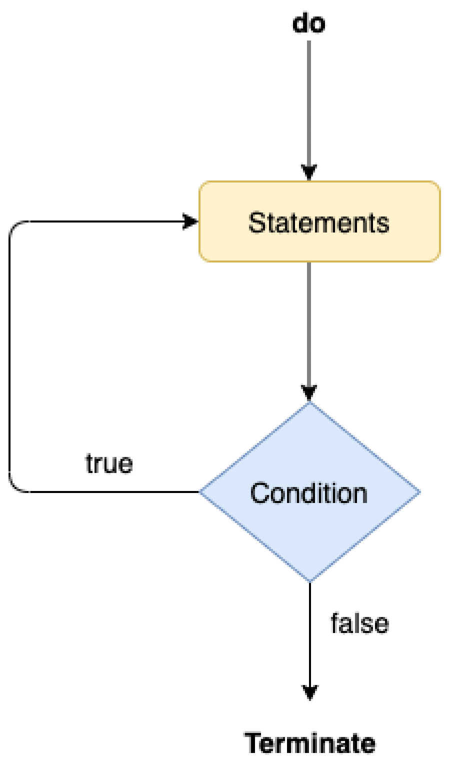
- A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.
- do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.

- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- It is important to note that the do-while loop will execute its statements atleast once before any condition is checked, and therefore is an example of exit control loop.

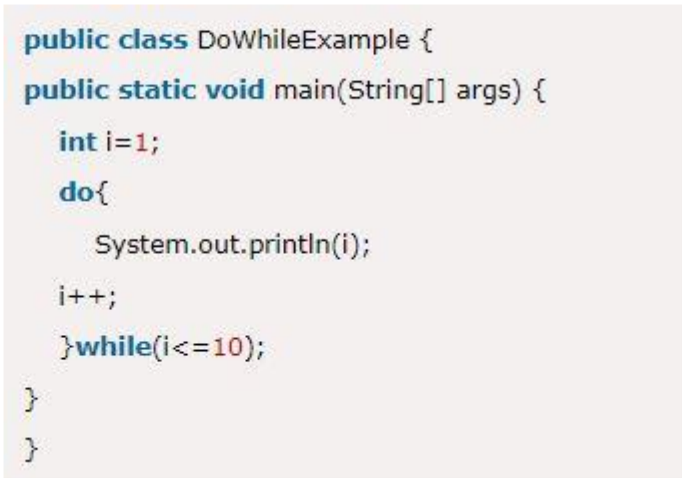
Syntax:

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

Flowchart:

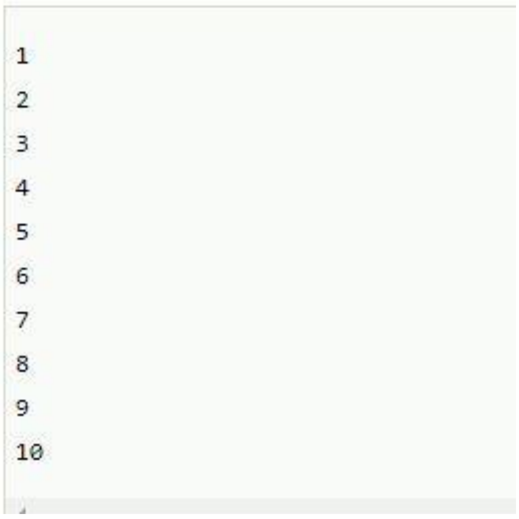


Example 1:



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

Output:

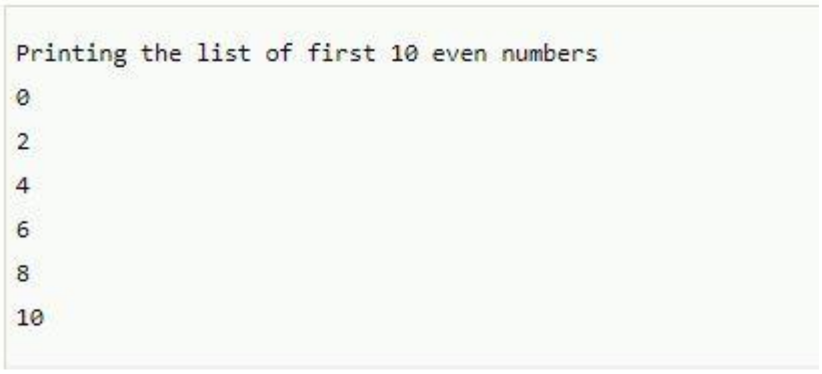


```
1
2
3
4
5
6
7
8
9
10
```

Example 2:

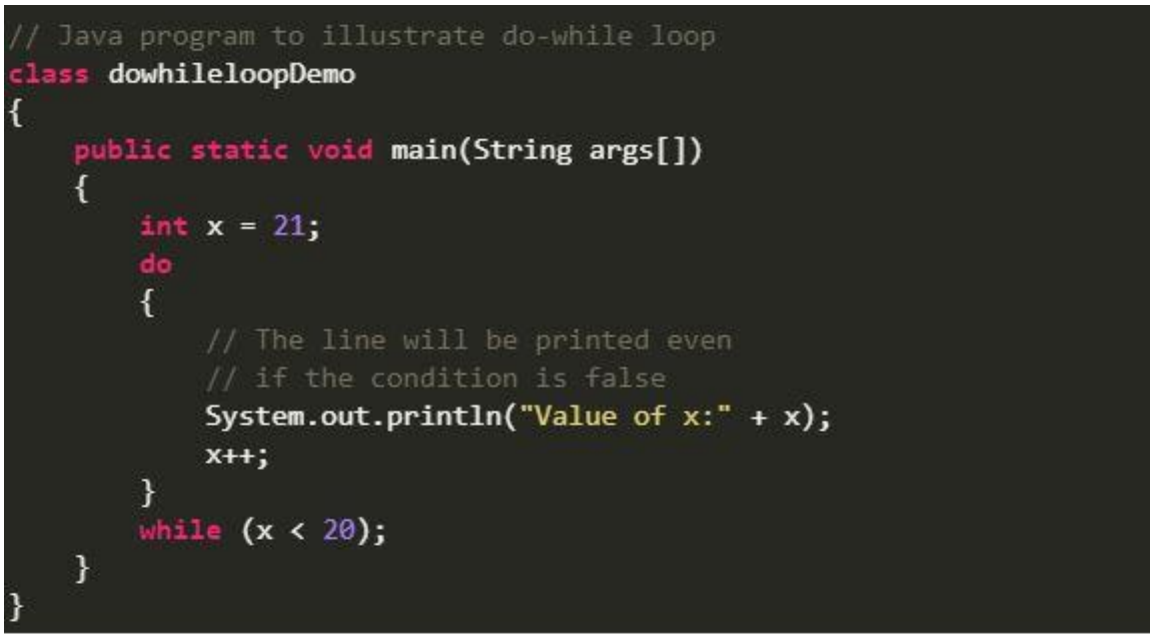
```java
public class Calculattion
{
        public static void main(String[] args)
        {
             int i = 0;
            System.out.println("Printing the list of first 10 even numbers \n");
           do
           {
               System.out.println(i);
               i = i + 2;
           }
while(i<=10);
}
}
```

Output:



```
Printing the list of first 10 even numbers
0
2
4
6
8
10
```

Example 3:



```
// Java program to illustrate do-while loop
class dowhileloopDemo
{
    public static void main(String args[])
    {
        int x = 21;
        do
        {
            // The line will be printed even
            // if the condition is false
            System.out.println("Value of x:" + x);
            x++;
        }
        while (x < 20);
    }
}
```

Output:

```
Value of x: 21
```

Jump Statement:

- Jump statements are used to transfer the control of the program to the specific statements.
- In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in java, i.e., break and continue.

1. Break Statement:

- 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.
- It is used to break the loop and switch statement.
- It breaks the current flow of the program at specified condition.

Syntax:

```
break;
```

Flow chart:

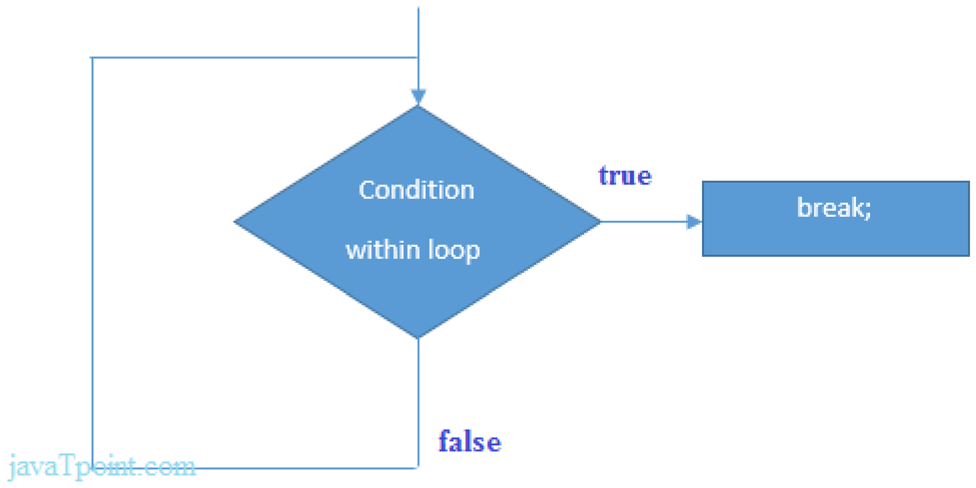
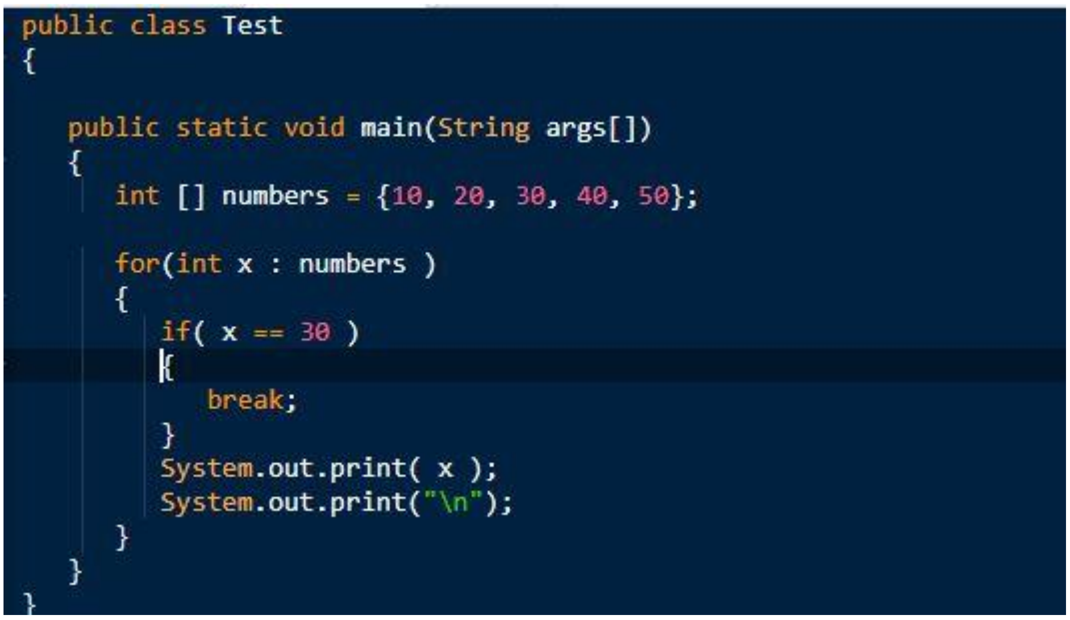


Figure: Flowchart of break statement

Example 1:



```
public class Test
{

    public static void main(String args[])
    {
        int [] numbers = {10, 20, 30, 40, 50};

        for(int x : numbers )
        {
            if( x == 30 )
            {
                break;
            }
            System.out.print( x );
            System.out.print("\n");
        }
    }
}
```

Output:

```
10
20
```

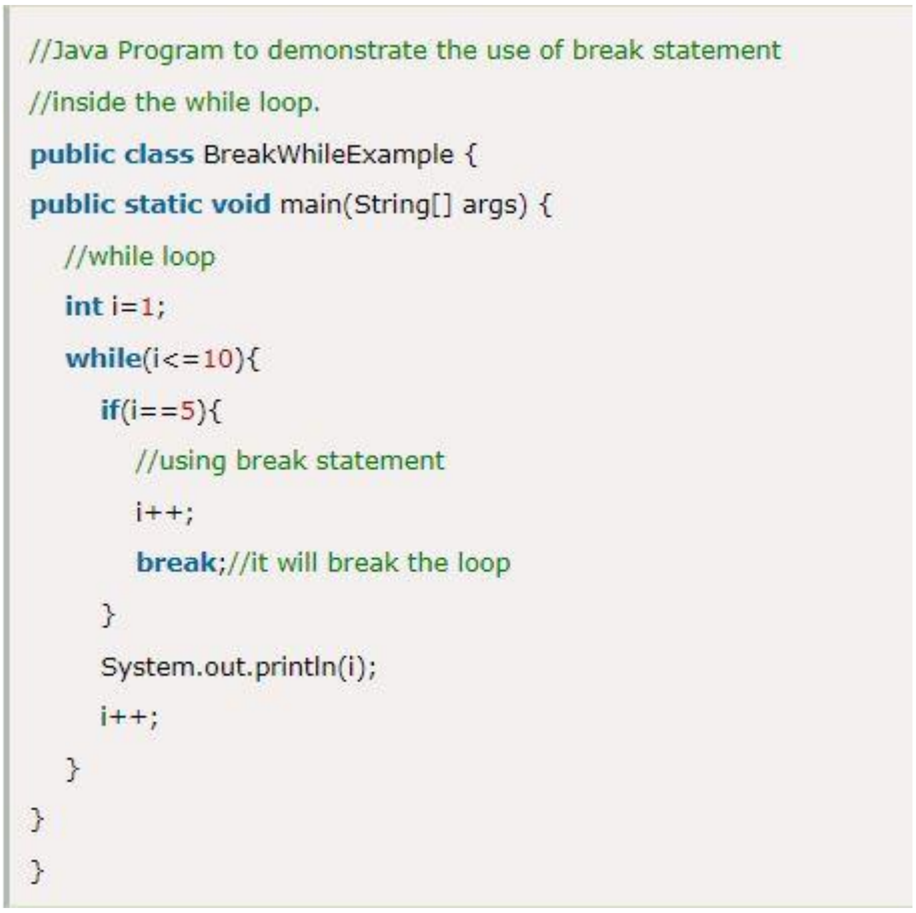
Example 2:

```
public class BreakExample
{
        public static void main(String[] args)
        {
            for(int i = 0; i<= 10; i++)
           {
              System.out.println(i);
             if(i==6)
             {
                break;
             }
           }
         }
}
```

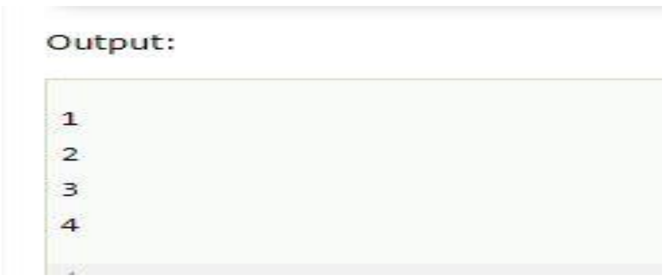
Output:

```
0
1
2
3
4
5
6
```

Example 3:



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



Output:

```
1
2
3
4
```

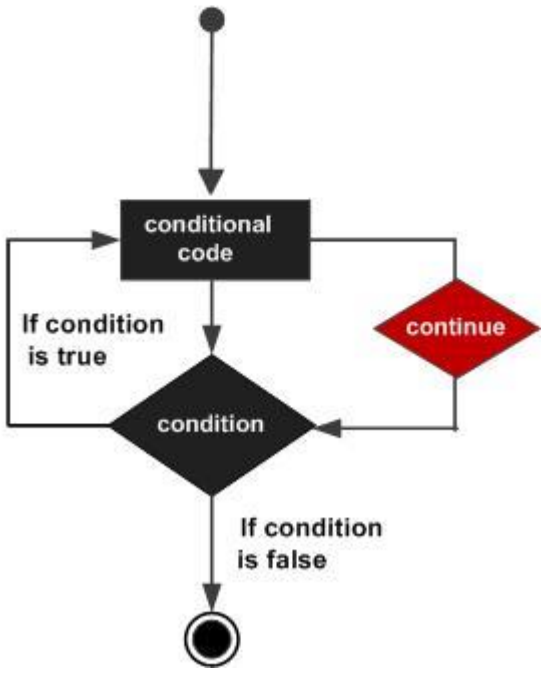
2. Continue Statements:

- the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.
- In a for loop, the continue keyword causes control to immediately jump to the update statement.
- In a while loop or do/while loop, control immediately jumps to the Boolean expression.
- It can be used with for loop or while loop.

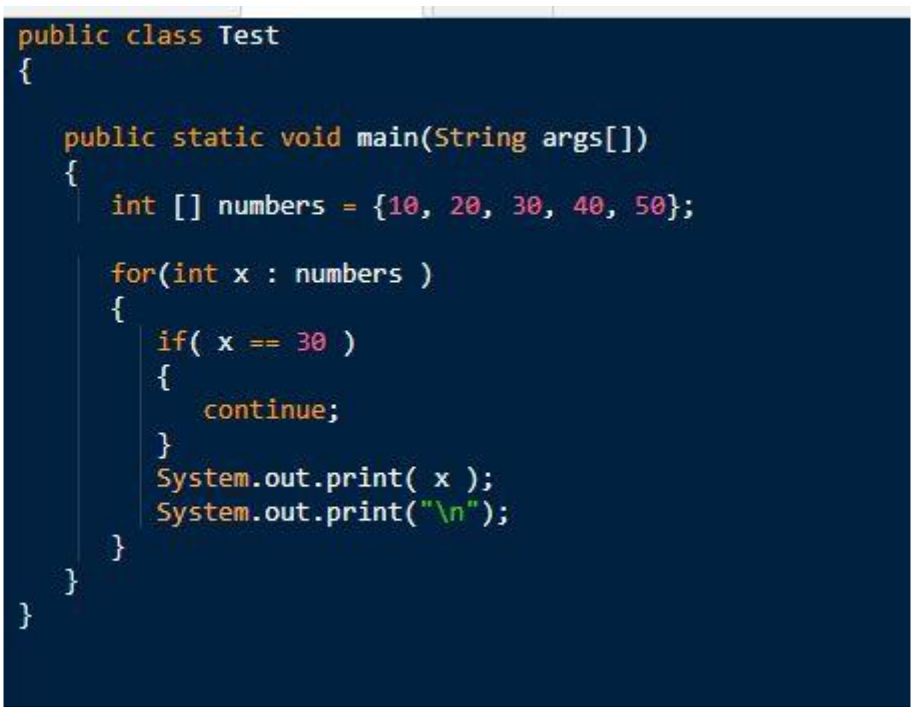
Syntax:

```
continue;
```

Flow chart:



Example 1:

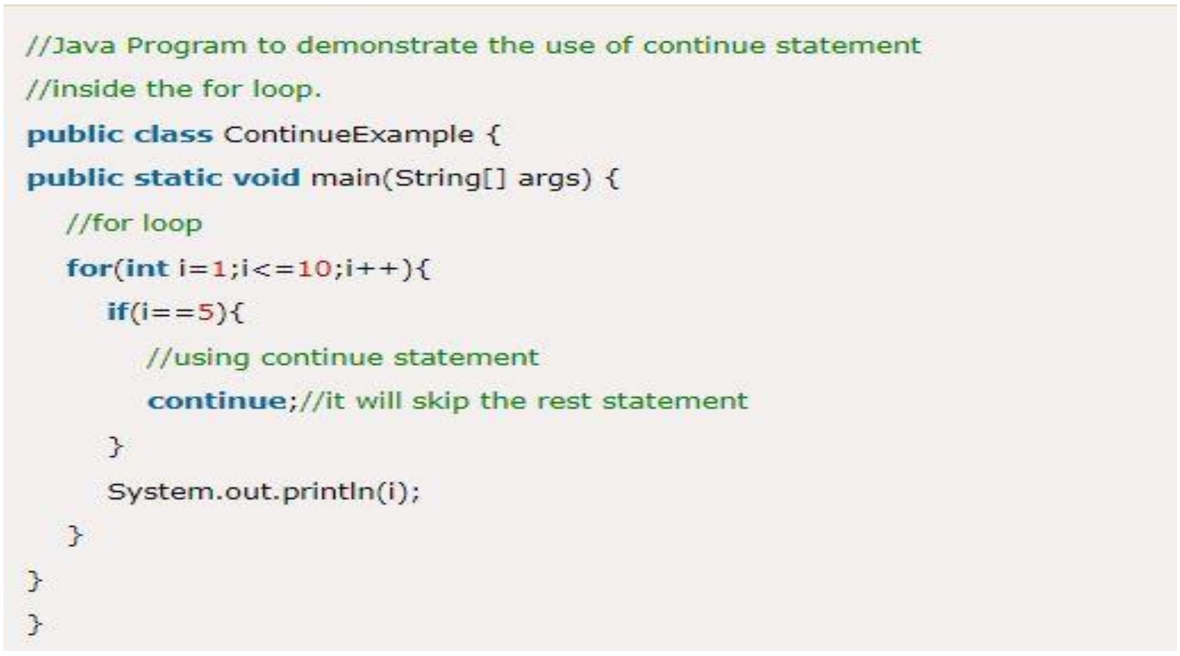


```java
public class Test
{

    public static void main(String args[])
    {
        int [] numbers = {10, 20, 30, 40, 50};

        for(int x : numbers )
        {
            if( x == 30 )
            {
                continue;
            }
            System.out.print( x );
            System.out.print("\n");
        }
    }
}
```

Output

```
10
20
40
50
```

Example 2:



```java
//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
```