Creating the variable

```
// decleration
datatype varible_name ;
```

Initialization

```
// initialization
varible_name = value;
```

Reinitialization

In reinitialization value will get **modified** with **new value**. If programmer is trying to print a value from a variable it always gives the **modified value**

Direct declaration with initialization

```
float mark1 = 50.0f;
float mark2 = 100.0f;
float mark3 = 80.5f;
```

Reinitialize or modification of initialized value

```
mark1 = 90.0f;
mark2 = 70.0f;
mark3 = 89.5f;
System.out.println(mark1);
System.out.println(mark2);
System.out.println(mark3);
```

> **Note:** duplicate variables are not allowed

Scope of a variable

- **Visibility** of a variable is known as scope of a variable
- Based on the scope of the variable , variable is categorized into **3 types**

1. Local variable

2. Static variable
3. Non-Static variable

Local variable

Any variable which is declared inside of the method block or any other block is known as **local variable**

Characteristics of Local variable

- These variables are not stored with default values
- These variables must be initialized before using it
- The scope of a local variable is only inside this specific block, Hence it cannot be used outside of the block

> **Example :**

```
{
// block 1
int a = 10;
System.out.println(a);
}
{
// block 2
int a = 20;
System.out.println(a);
}
```

- Block 1 will print **10** and Block 2 will print **20** ,
-

because here the 2 a variables are local variable which can only act inside the block.

> **Note :**

- We can create variables with same name into different local blocks .

Datatype	Default Value	Size
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0	8 byte

Datatype	Default Value	Size
float	0.0f/F	2 byte
double	0.0d/D	4 byte
char	space (or) `\u000'	2 byte
Boolean	false	1 bit

Operators

Operators are the **predefined symbols** which is used to perform some specific tasks on given **operands**

Operands

Operands are the data given to the operator. Operands are 3 types

- **unary operator** - which works on **1 operand**
- **Binary operator** - which works on **2 operands**
- **Ternary operator** - which works on **3 or more operands**

Based on types of operators, operators are classified as

1. Arithmetic operator
2. Assignment operator
3. Relational operator
4. Logical operator
5. Increment/Decrement operator
6. Conditional operator

1) Arithmetic operator

Arithmetic operator is used to perform **arithmetic operation** on given operand. **Primitive values** should be used for arithmetic operator.

Arithmetic operations

```
    int a = 10;
    int b = 20;
```

1. Addition (+)

```
    System.out.println(a+b) // 30
```

2. Subtraction (-)

```
    System.out.println(a-b) // -20
```

3. Multiplication (*)

```
     System.out.println(a*b) // 300
```

4. Division (/)

```
     System.out.println(a/b) // 0
     System.out.println(11/2) // 5
     System.out.println(3/2) // 1
     System.out.println(56/7) // 8
```

5. Modulus (%)

```
     System.out.println(a%b) // 10
     System.out.println(11%2) // 1
     System.out.println(3%2) // 1
     System.out.println(3%7) // 3
```

Assignment operator

- Assignment operator is used to assign a value a value to a variable
- **Assignment - =**

```
    int a = 10;
    Sytem.out.println(a) //10
```

- **Addition assignment - +=**

```
    a += 5;
    //a = a + 5;
    Sytem.out.println(a) //5
```

- **Subtraction assignment - -=**

```
    a -= 5;
    // a = a - 5;
    Sytem.out.println(a) //5
```

- **Multiplication assignment - *=**

```
a *= 5;
//a = a * 5;
Sytem.out.println(a) //50
```

- Division - /=

```
a /= 5;
// a = a/5;
Sytem.out.println(a) //2
```

- Modulus - %=

```
a %= 5;
// a = a % 5
Sytem.out.println(a) //0
```

Relational operator

- It is used to check the **relation** between **2 operands** . The **return type** of the this operator is **Boolean**
- The operators are

```
int a = 3;
int b = 2;
```

1. **Equality** - ==: Its is used to compare the operands are same .

```
boolean c = a==b;
System.out.println(c); // false
int x = 10;
int y = 10;
boolean z = x==y;
System.out.println(c); // true
```

2. **Less than** - <: Its checks that the left side value is lesser than the right side value

```
boolean d = a<b;
System.out.println(d); //false
```

```

3. **Greater than** - >: Its checks that the left side value is greater than the right side value.

```
 boolean e = a>b;
```
```

```
    System.out.println(e); //true
```

4. **Less than equals to** - <= Its checks that the left side value is less than or equals the right side value.

```
    boolean f <= a>b
    System.out.println //false
```

6. **Increment/Decrement** : there are 2 increment type they are 1) Pre-increment 2) Post-increment

```
int m = 5;
int n = 3;
int res = ++m + m++;
System.out.println(res);// 12
System.out.println(m);// 7
System.out.println(n);// 3
```

> Example :

```
package  operators;

public  class  IncDec {
public  static  void  main(String[] args) {
int  x  =  5;
int  y  =  10;
int  z  =  ++x  +  x  +  --y  +  y++  +  y;
z++;
System.out.println(x); //6
System.out.println(y); //10
System.out.println(z); //41
}
}
```

5) Logical Operator:

- Its is used to perform **logical operation** on given **Boolean operands**.
- Its gives **Boolean result** The Logical operators are,

1. Logical AND (&&)
2. Logical OR (||)
3. Logical Not (!)

> OR table

O1	O2	res
F	F	F
F	T	T
T	F	T
T	T	T

> AND table

O1	O2	res
F	F	F
F	T	F
T	F	F
T	T	T

> Example:

```
    int a = 2;
    int b = 3;
    boolean c = a>b || a<b
    System.out.println(c); // T (true)
```

> **Note**: In logical or operator if 1st operand is true (T) then compiler never checks the other operand

```
package  operators;

public  class  OrOperator {
    public  static  void  main(String[] args) {
        int  a  =  2;
        int  b  =  3;
        boolean  c  = ( a++  <=  2  || b++  >=  3);
        System.out.println(c); // T (true)
        System.out.println(a); // 3
        System.out.println(b); // 3
    }
}
```

Logical AND

> **Example**

```
package  operators;

public  class  IncDec {
    public  static  void  main(String[] args) {
        int  x  =  10;
        int  y  =  20;
        boolean  z  = ( ++x  !=  y  &&  y++  ==  20);
        System.out.println(z); // T (true)
        System.out.println(x); // 11
        System.out.println(y); // 21
    }
}
```

> **Note:** In logical AND operator if 1 operand is false then compiler never checks the other operand

Logical NOT

Its will give the Boolean value of the variable

```
package  operators;

public  class  IncDec {
    public  static  void  main(String[] args) {
        boolean  x  =  true;
        boolean  z  =  !x;
        System.out.println(z); // F (false)
        System.out.println(x); // T (True)
    }
}
```

Conditional Statement

Syntax

```
    condition ? statement1 : statement2 ;
```

Eg:

```
package  operators;

public  class  ConditionalOp {
    public  static  void  main(String[] args) {
        int  x  =  20;
        int  y  =  20;
        String  result  = (x  !=  y  ?  "X is not equal to Y":" X is equal to Y");
        System.out.println(result); // X is equal to Y
    }
}
```

> write a program to check the given number is even or odd. ?

```
package  operators;

public  class  ConditionalOp {
    public  static  void  main(String[] args) {
        int  x  =  20;
        String  result  = (x  %  2  ==  0)?("X is even number"):("X is not even number");

        System.out.println(result) // X is even number;
    }
}
```

> write a program to check the largest of three numbers

```
public  class  LargestOfThree {
    public  static  void  main(String[] args) {
        int  a  =15;
        int  b  =100;
        int  c  =  20;
        String  res  = (a>b?
        // if a is greater than b
        (a>c?
        // if a is greater than c
        ("A is greater")
        :
        // if c is greater than a
        ("C is greater"))
        :
        // if b is greater than a
        (b>c?
        // if b is greater than c
        ("B is greater")
        :
        // if c is greater than b
        ("C is greater"))
        );
        System.out.println(res); // B is greater
    }
}
```

> **Note:** For conditional statement , statement1 and statement2 must be of same type otherwise we will get compile time error

Concatenation

Combination of **2 or more strings** together to form a new string is known as concatenation.

1. Number + Number + is acting as a addition operation

```
public class Example {
    public static void main(String[] args) {
        System.out.println(10 + 10); // 20
    }
}
```

2. Number + Character character gets convert to **ASCII value** and gets added together

```
public class Example {
    public static void main(String[] args) {
        System.out.println(10 + 'a'); //107
    }
}
```

3. Character + Character character gets convert to **ASCII value** and gets added together

```
public class Example {
    public static void main(String[] args) {
        System.out.println(10 + true); // Not possible
    }
}
```

4. Number + Boolean This method is **Not Possible**

```
public class Example {
    public static void main(String[] args) {
        System.out.println(10 + true); // Not possible
    }
}
```

5. Number + String + is Acting as a concatenation operator

```
public class Example {
    public static void main(String[] args) {
        System.out.println(10 + "abc"); //10abc
        System.out.println("abc" + 10); //abc10
        System.out.println(10 + 10 + "abc"); //20abc
        System.out.println(10 + 'a' + "abc"); //107abc
        System.out.println(10 + "-abc-" + "-cde-"); //10-abc--cde-
```

```java
        System.out.println(10  +  "-abc-"  +  4  +  20); //10-abc-420
        System.out.println(10  +  "-abc-"  +  'a'); //10-abc-a
        System.out.println(10  +  "-abc-"  +  true); //10-abc-true
        // System.out.println(10 + true + "-abc-"); //compiler error
        // System.out.println(10 + "-abc-" +20 - 30); //compiler error
        System.out.println(10  +  "-abc-"  +(20  -  30)); //10-abc--10
    }
}
```