

1) WAP to enter two number and add using command line argument

```
class add
{
    public static void main(String args[])
    {
        int a,b,sum;
        a=Integer.parseInt(args[0]);
        b=Integer.parseInt(args[1]);
        sum=a+b;
        System.out.println("sum=" +sum);
    }
}
```

java add 10 20

Addition two float values

```
a=Float.valueOf(args[0]).floatValue();
b=Float.valueOf(args[1]).floatValue();
```

Addition of two double values

```
a=Double.parseDouble(args[0]);
b=Double.parseDouble(args[1]);
```

WAP to addition of n numbers using Command-line arguments

```
class sumn
{
    public static void main(String args[])
    {
        int i,sum=0;
        for(i=0;i<args.length;i++)
        {
            sum=sum+Integer.parseInt(args[i]);
        }
        System.out.println("sum=" +sum);
    }
}
```

java sumn 10 20 30 40 50

Variables

A Variable is an identifier that denotes a storage location used to store a data value. Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during the execution of the program. A Variable name can be chosen by the programmer in a meaningful way.

Variable names may consist of alphabets, digits, the underscore(_) and dollar characters.

1. They must not begin with a digit.

2. UpperCase and lowercase are distinct. This means that the variable Total is not the same as total or TOTAL.
3. It should not be a keyword.
4. White space is not allowed.
5. Variable names can be of any length.

In Java, Variables are the names of storage locations. Variable declaration does three things :

1. It tells the compiler what the variable name is.
2. It specifies what type of data the variable will hold.
3. The place of declaration (in the program) decides the scope of the variable.

Syntax :

Type variable1, variable2, variableN;

Ex :- int count;
float x,y;
double pi;
byte b;
char c1,c2,c3;

Giving Values to Variables

A Variable must be given a value after it has been declared but before it is used in an expression. This can be achieved in two ways :

1. By using an assignment statement
2. By using a read statement.

1. Assignment Statement

A Simple method of giving value to a variable is through the assignment statement as follows :

variableName=value;

For example :

initial=0;
finalvalue=100;
yes='x';

We can also string assignment expression as follows :

x=y=z=0;

It is also possible to assign a value to a variable at the time of its declaration.

Example : int a=50;

The process of giving initial values to variables is known as the initialization.

Read Statement

We may also give value to variables interactively through the keyboard using the readLine() method.

```
import java.io.*;  
class read
```

```

{
public static void main(String args[]) throws IOException
{
    int a,b,sum;
    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Enter the first number :");
    a=Integer.parseInt(br.readLine());
    System.out.println("Enter the second number :");
    b=Integer.parseInt(br.readLine());
    sum=a+b;
    System.out.println("sum=" +sum);
}
}

```

Scope of Variables

Java variables are actually classified into three kinds :

1. instance variables,
2. class variables, and
3. local variables

1. **instance variables** -> Instance variables are created when the objects are instantiated and therefore they are associated with the objects. They take different values for each object.
2. **class variables**-> class variables are global to a class and belong to the entire set of objects that class creates. Only one memory location is created for each class variables.
3. **local variables** -> Variables declared and used inside methods are called local variables. They are called so because they are not available for use outside the method definition. Local variables can also be declared inside program blocks that are defined between an opening brace { and a closing brace }. These variables are visible to the program only from the beginning of its program block to the end of the program block. When the program control leaves a block, all the variables in the block will cease to exist. The area of the program where the variable is accessible (i.e, usable) is called its scope.

Symbolic Constants

A constant is declared as follows :

Syntax :- final type symbolic-name=value;

Examples :

```

final int STRENGTH=100;
final int PASS_MARK=50;
final float PI=3.14159;

```

Note :

1. Symbolic names take the same form as variable names. But, they are written in CAPITALS to visually distinguish them from normal variable names.

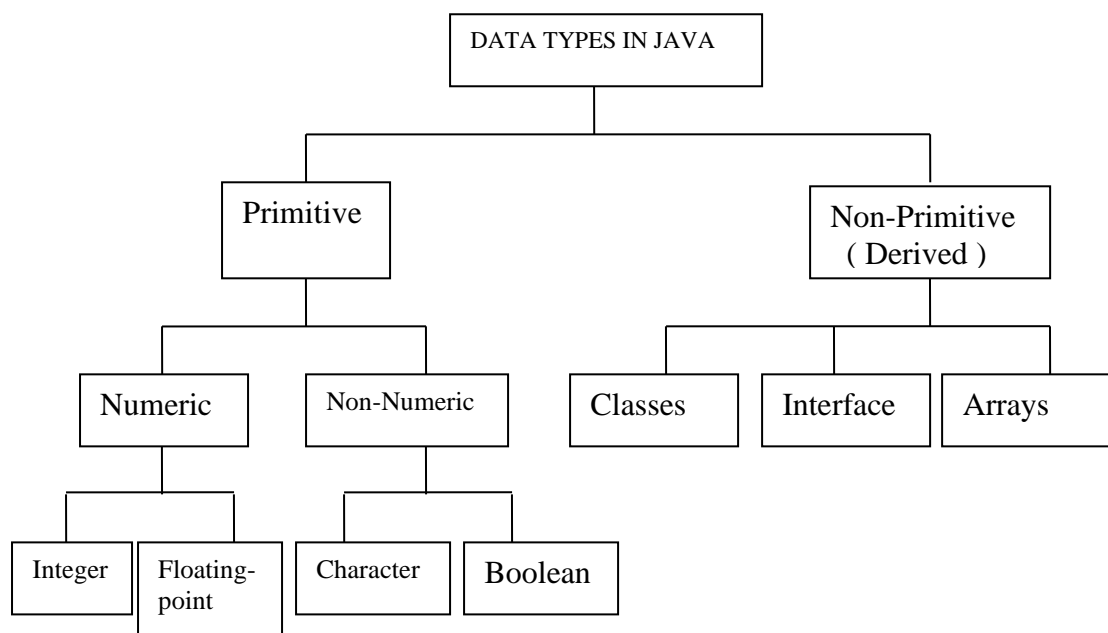
2. After declaration of symbolic constants, they should not be assigned any other value within the program by using an assignment statement. For example, `STRENGTH=200;` is illegal.
3. Symbolic constants are declared for types.

Data Types

Data types specify the size and type of values that can be stored. Java language is rich in its data types. The variety of data types available allow the programmer to select the type appropriate to the needs of the application.

Java supports two classes of data types :-

1. Primitive data types
2. Non-Primitive(Derived) data types



1. Integer Types

Integer types can hold whole numbers such as 123,-96. Java supports four types of integers. They are byte, short, int, and long. Java does not support the concept of unsigned types and therefore all Java values are signed meaning they can be positive or negative.

Type	Size	Minimum value	Maximum value
byte	One byte	-128	+127
short	Two byte	-32,768	+32,767
int	Four bytes	-2,147,483,648	+2,147,483,647
long	Eight bytes

2. Floating Point Types

It holds fractional parts such as 27.59 and -1.375 (known as floating point constants). There are two kinds of floating point storage in Java.

The float type values are single-precision numbers while the double types represent double-precision numbers.

Floating point numbers are treated as double-precision quantities. To force them to be in single-precision mode, we must append f or F to the numbers.

Example : 1.23f
7.569f

Type	Size	Minimum value	Maximum value
float	4 bytes	3.4e-038	3.4e+038
double	8 bytes	1.7e-308	1.7e+308

Double-precision types are used when we need greater precision in storage of floating point numbers. All mathematical functions, such as sin, cos and sqrt return **double** type values.

Character Type

In order to store character constants in memory, Java provides character data type called char. It holds 2 bytes of memory.

Boolean Type

Boolean type is used when we want to test a particular condition during execution of the program. There are only two values that a boolean type can take : true or false. Boolean type is denoted by the keyword **boolean** and uses only 1 bit of storage.

Type Casting

To store a value of one type into a variable of another type. In such situations, we must cast the value to be stored by proceeding it with the type name in parentheses.

Syntax : type variable1=(type) variable2;

The process of converting one data type to another is called casting.

Examples :

```
int m=50;
byte n=(byte) m;
long count=(long)m;
```

Casting into a smaller type may result in a loss of data. Similarly, the **float** and **double** can be cast to any other type except boolean. Casting a floating point value to an integer will result in a loss of the fractional part.

Automatic Conversion

For some types, it is possible to assign a value of one type to a variable of a different type without a cast. Java does the conversion of the assigned value automatically. This is known as automatic type conversion. Automatic type conversion is possible only if the destination type has enough precision to store the source value. For example,

```
byte b=75;
int a=b;
```

The process of assigning a smaller type to a larger one is known as widening or promotion and that of assigning a larger type to a smaller one is known as narrowing. Narrowing may result in loss of information.

Decision Statements

The decision statement decides the statement to be executed after the success or failure of a given condition. The decision-making statement checks the given condition and then executes its sub-block.

Java language possesses decision-making capabilities supporting the following statements :-

1. if statement
2. switch statement
3. conditional operator statement

These statements are popularly known as decision –making statements. Since these statements control the flow of executing, they are also known as control statements.

- a) The if statement
- b) The if-else statement
- c) Nested if-else statement
- d) The if-else-if ladder statement

a) The if statement

Java uses the keyword if to execute a set of command lines or one command line when the logical condition is true. It has only one option. The set of command lines or command lines are executed only when the logical condition is true.

Syntax :

```
if(condition)
    statements;
e.g,
```

WAP to check whether the entered number is less than 10? if yes, display the same.

```
class num
{
    public static void main(String args[])
    {
        int v;
        if(v<10)
        {
            System.out.println("Number entered is less than 10");
        }
    }
}
```

2. The if..else statement

The if..else statement takes care of true as well as false conditions. It has two blocks. One block is for if and it is executed when the condition is true. The other block is of else and it is executed when the condition is false. The else statement cannot be used without if. No multiple else statements are allowed with one if.

Syntax :

```
if(condition)
    statement1;
else
    statement2;
```

WAP to enter number and check no. is even or odd.

```
import java.io.*;
class odd
{
    public static void main() throws IOException
    {
        int num;
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter number :");
        num=Integer.parseInt(br.readLine());
        if(num%2==0)
            System.out.println("even number");
        else
            System.out.println("odd number");
    }
}
```

Nested if..else

```
if(condition)
    statement;
else
{
    if(condition)
        statement;
    else
    {
        statement;
    }
}
```

WAP to enter three numbers and find largest.

```
import java.io.*;
```

```

class large
{
public static void main(String args[]) throws IOException
{
    int a,b,c;
    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Enter three number :");
    a=Integer.parseInt(br.readLine());
    b=Integer.parseInt(br.readLine());
    c=Integer.parseInt(br.readLine());
    if(a>b)
        if(a>c)
        {
            System.out.println("a is greater");
        }
    else
    {
        System.out.println("c is greater");
    }
    else if(b>c)
    {
        System.out.println("b is greater");
    }
    else
    {
        System.out.println("c is greater");
    }
}
}

```

The if..else if ladder

Syntax :

```

if(condition1)
    statement;
else if(condition2)
    statement;
else if(condition3)
    statement;
.....
else
    statement;
e.g,

```

WAP to enter five subjects of marks and calculate sum and average.

If avg>=60 print "First division"

Avg>=45 and avg<60 "Second division"

Avg>=30 and avg<45 "Third division"

Else fail


```

import java.io.*;
class first
{
    public static void main(String args[]) throws IOException
    {
        int phy,che,math,bio,eng;
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter Five subject of marks :");
        phy=Integer.parseInt(br.readLine());
        che=Integer.parseInt(br.readLine());
        math=Integer.parseInt(br.readLine());
        bio=Integer.parseInt(br.readLine());
        eng=Integer.parseInt(br.readLine());
        float sum,avg;
        sum=phy+che+math+bio+eng;
        avg=sum/5;
        if(avg>=60)
            System.out.println("First division");
        else if(avg>=45 && avg<60)
            System.out.println("Second division");
        else if(avg>=30 && avg<45)
            System.out.println("Third division");
        else
            System.out.println("Fail");
    }
}

```

The break statement

The keyword break allows the programmers to terminate the loop. The break skips from the loop or block in which it is defined. The control then automatically goes to the first statement after the loop or block. The break can be associated with all conditional statements.

We can also use break statements in the nested loops. If we use break statement in the innermost loop then the control of the program is terminated only from the innermost loop.

The continue statement

The continue statement is exactly opposite to break. The continue statement is used for continuing next iteration of loop statements. When it occurs in the loop it does not terminate, but it skips the statements after this statement. It is useful when we want to continue the program without executing any part of the program.

Difference between break and continue

break

- 1) exits from current block or loop
- 2) control passes to next statement
- 3) terminates the program

continue

- loop takes next iteration
- control passes at the beginning of loop.
- never terminates the program.