**Variable**

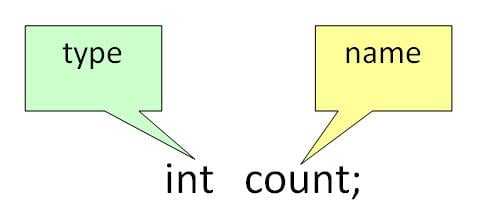
**Variable in Java** is a data container that stores the data values during Java program execution. Every variable is assigned data type which designates the type and quantity of value it can hold. Variable is a memory location name of the data. The Java variables have mainly three types : Local, Instance and Static.

In order to use a variable in a program you to need to perform 2 steps

1. Variable Declaration
2. Variable Initialization

**Variable Declaration:**

To declare a variable, you must specify the data type & give the variable a unique name.



Examples of other Valid Declarations are

int a,b,c;

float pi;

double d;

char a;

**Variable Initialization:**

To initialize a variable, you must assign it a valid value.

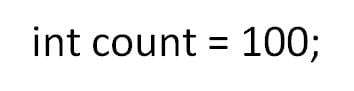
Example of other Valid Initializations are

pi =3.14f;

do =20.22d;

a=’v’;

You can combine variable declaration and initialization.

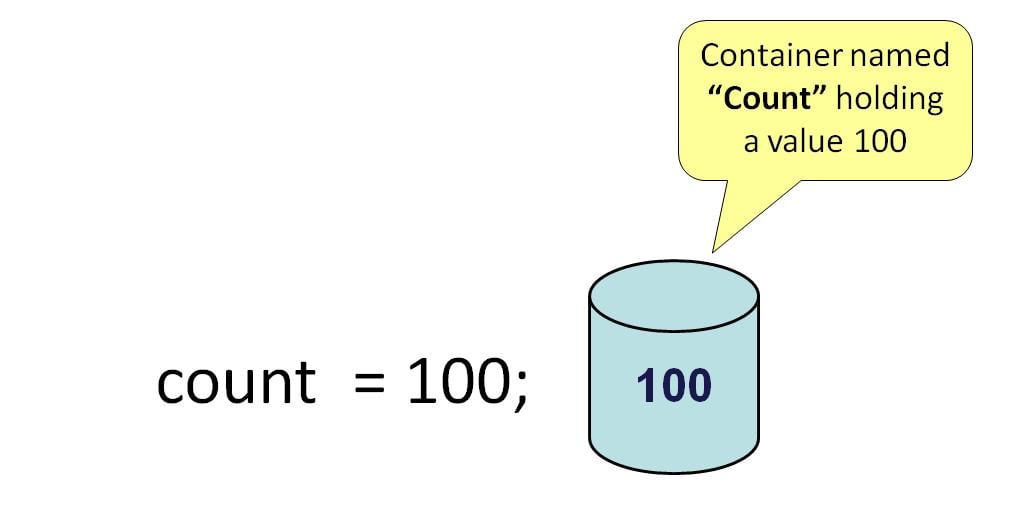


Example :

int a=2,b=4,c=6;

float pi=3.14f;

double do=20.22d;

char a=’v’;

## Types of variables

In Java, there are three types of variables:

1. Local Variables
2. Instance Variables
3. Static Variables

### 1) Local Variables

Local Variables are a variable that are declared inside the body of a method.

### 2) Instance Variables

Instance variables are defined without the STATIC keyword .They are defined Outside a method declaration. They are Object specific and are known as instance variables.

### 3) Static Variables

Static variables are initialized only once, at the start of the program execution. These variables should be initialized first, before the initialization of any instance variables.

## Example: Types of Variables in Java

class Variables {

static int a = 1; //static variable

int data = 99; //instance variable

void method() {

int b = 90; //local variable

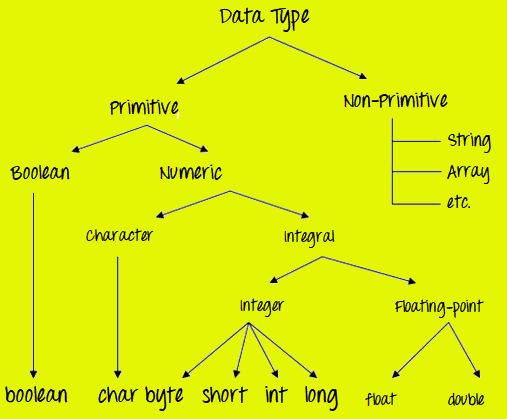
}

}

## What is Data Types in Java?

**Data Types in Java** are defined as specifiers that allocate different sizes and types of values that can be stored in the variable or an identifier. Java has a rich set of data types. Data types in Java can be divided into two parts :

1. **Primitive Data Types** :- which include integer, character, boolean, and float
2. **Non-primitive Data Types** :- which include classes, arrays and interfaces.



### Primitive Data Types

Primitive Data Types are predefined and available within the Java language. Primitive values do not share state with other primitive values.

There are 8 primitive types: byte, short, int, long, char, float, double, and boolean

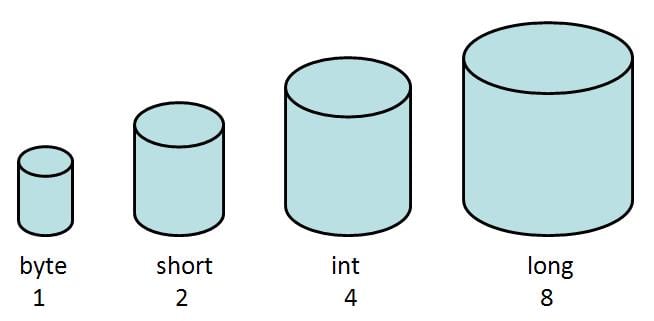
**Integer data types**

byte (1 byte)

short (2 bytes)

int (4 bytes)

long (8 bytes)



**Floating Data Type**

float (4 bytes)

double (8 bytes)

**Textual Data Type**

char (2 bytes)

**Logical**

boolean (1 byte) (true/false)

**package** com.newjavacourse;

**public** **class** Student {

/\*

\* int z=30;//instance variable heap memory static int b=30; // class area

\* static String t= "peit";

\*/

**float** f1=40;

**public** **static** **void** main(String[] args) {

/\*int a = 10;//local variable stack

int x =20;

Student chandu = new Student();

System.out.println(a);

System.out.println(x);

System.out.println(chandu.z);

System.out.println(Student.b);

System.out.println(Student.t);\*/

Student chandu = **new** Student();

**boolean** b=**false**;

System.***out***.println(b);

**int** i=10;

System.***out***.println(i);

**long** l=1000;

System.***out***.println(l);

**float** f =10.5f;

System.***out***.println(f);

**double** d=11.5;

System.***out***.println(d);

System.***out***.println(chandu.f1);

/\*primitive -

\* boolean,int,float,char,long,double

\*

\*

\* non primitive-

\*

\* class,inerface,arrays

\* class---blue print,sketch,version

\* object--instance of a class

\*

\* properties(fields) behaviour(methds)-state behaviour

\*

\*

\*

\*

\*

\*

\*

\*

\*/

}

}

Class and Object

Java is an object-oriented programming language. The core concept of the object-oriented approach is to break complex problems into smaller objects.

An object is any entity that has a **state** and **behavior**. For example, a bicycle is an object. It has

* **States**: idle, first gear, etc
* **Behaviors**: braking, accelerating, etc.

**Java Class**

A class is a blueprint for the object. Before we create an object, we first need to define the class.

We can think of the class as a prototype of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object.

Since many houses can be made from the same description, we can create many objects from a class.

**Create a class in Java:**

We can create a class in Java using the class keyword. For example,

class ClassName {

// fields

// methods

}

Here, fields (variables) and methods represent the **state** and **behavior** of the object respectively.

* fields are used to store data
* methods are used to perform some operations

For our bicycle object, we can create the class as

class Bicycle {

// state or field

private int gear = 5;

// behavior or method

public void braking() {

System.out.println("Working of Braking");

}

}

In the above example, we have created a class named Bicycle. It contains a field named gear and a method named braking().

Here, Bicycle is a prototype. Now, we can create any number of bicycles using the prototype. And, all the bicycles will share the fields and methods of the prototype.

We have used keywords private and public. These are known as access modifiers.

## Java Objects:

An object is called an instance of a class. For example, suppose Bicycle is a class then MountainBicycle, SportsBicycle, TouringBicycle, etc can be considered as objects of the class.

### Creating an Object in Java

Here is how we can create an object of a class.

### Creating an Object in Java

Here is how we can create an object of a class.

className object = new className();

// for Bicycle class

Bicycle sportsBicycle = new Bicycle();

Bicycle touringBicycle = new Bicycle();

We have used the new keyword along with the constructor of the class to create an object. Constructors are similar to methods and have the same name as the class. For example, Bicycle() is the constructor of the Bicycle class.

Here, sportsBicycle and touringBicycle are the names of objects. We can use them to access fields and methods of the class.

As you can see, we have created two objects of the class. We can create multiple objects of a single class in Java.

Examples:

**class** Bicycle {

// field of class

**int** gear = 5;

**int** wheels= 4;

// method of class

**void** braking() {

System.***out***.println("Sprots byke is running");

}

**public** **static** **void** main(String[] args) {

// create object

Bicycle sportsBicycle = **new** Bicycle();

Bicycle sportsCar = **new** Bicycle();

// access field and method

System.***out***.println(sportsBicycle.gear);

System.***out***.println(sportsCar.gear);

sportsBicycle.braking();

System.***out***.println("The wheels of Sports car is: " +sportsCar.wheels);

}

}

Examples:

**package** com.newjavacourse;

**public** **class** Car {

**int** gear=5;

**public** **void** run() {

System.***out***.println("bmw car is running");

}

**public** **void** run1() {

System.***out***.println("mg car is running");

}

**public** **void** run2() {

System.***out***.println("tuv car is running");

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Car bmw = **new** Car();

System.***out***.println("bmw car has "+bmw.gear + " gears");

bmw.run();

Car mg = **new** Car();

System.***out***.println("bmw car has "+mg.gear + " gears");

mg.run1();

Car tuv = **new** Car();

System.***out***.println(tuv.gear);

tuv.run2();

}

}