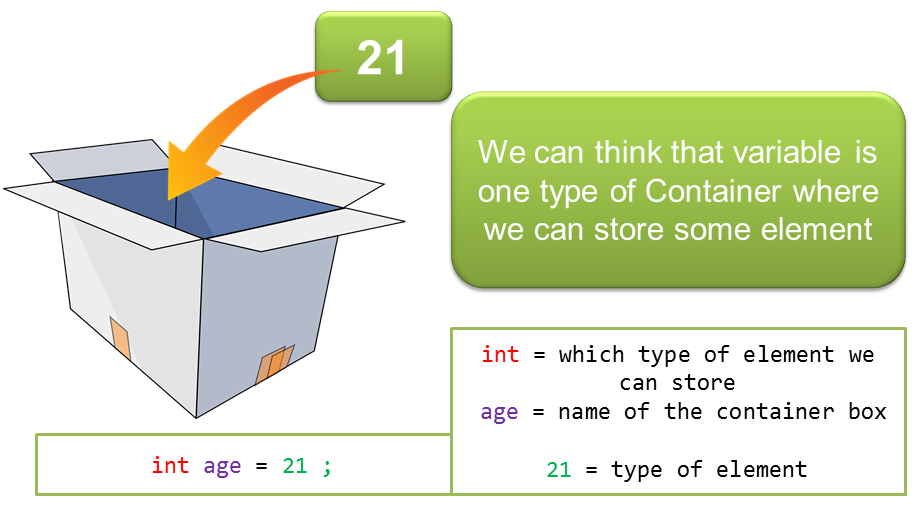
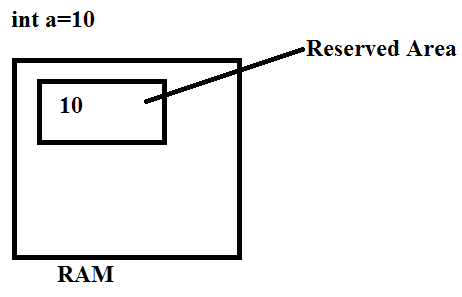
**Java Variables**

A variable is a container which holds the value while the java program is executed. A variable is assigned with a data type.

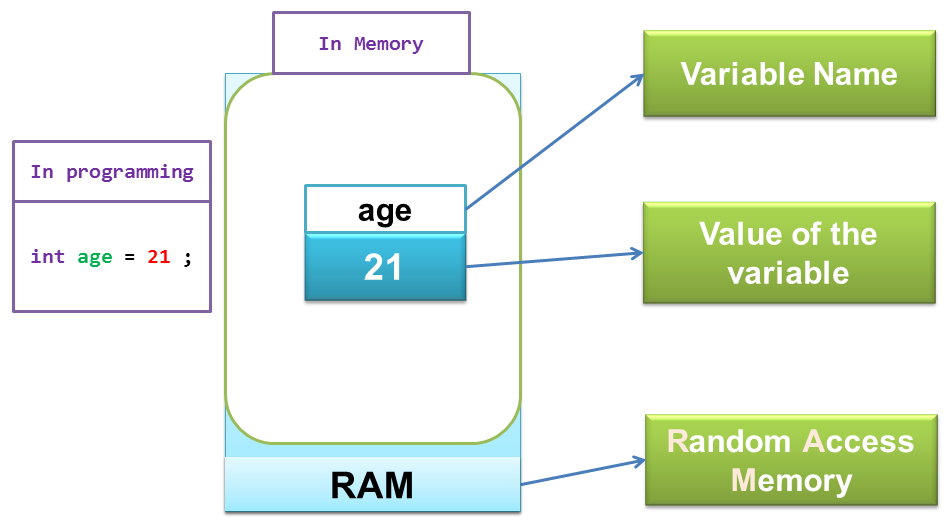


* Variables are the identifier of the memory location, which used to save data temporarily for later use in the program.

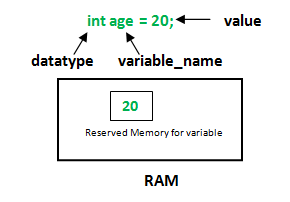


* During execution of a program, values can be stored in a variable, and the stored value can be changed.
* Variable is name of reserved area allocated in memory (name of memory location).
* It is the basic unit of storage in a program.
* A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.

**Note**: In Java, all the variables must be declared before use, otherwise we will get compile time error.



**How to declare variables?**

**[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/Variables-in-Java.png)**

**datatype**: Type of data that can be stored in this variable.  
**variable\_name**: Name given to the variable.  
**value**: It is the initial value stored in the variable.

**Rules to declare a Variable**

* Every variable name should start with either alphabets or underscore ( \_ ) or dollar ( $ ) symbol.
* No space is allowed in the variable declarations.

Ex: int num ber--invalid

* Except underscore ( \_ ) no special symbol are allowed in the middle of variable declaration
* Variable name always should exist in the left hand side of assignment operators.
* Maximum length of variable is 64 characters.
* No keywords should access variable name.

**Note:** Actually a variable also can start with ¥,¢, or any other currency sign.

**Ex:**

class Sum{

public static void main(String[] args) {

int \_a, ¢b, ¥c, $d, result;

\_a=10;

¢b=20;

¥c=30;

$d=40;

result=\_a+¢b+¥c+$d;

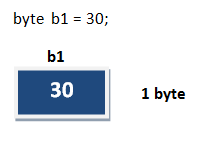
System.out.println("Sum is :" +result);

}

}

**Variable initialization:**

It is the process of storing user defined values at the time of allocation of memory space.



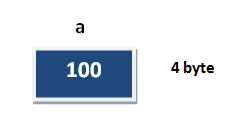
**Variable assignment:**

Value is assigned to a variable if that is already declared or initialized.

**Syntax**

Variable\_Name = value

**int** a = 100;



**Syntax**

**int** a= 100;

**int** b;

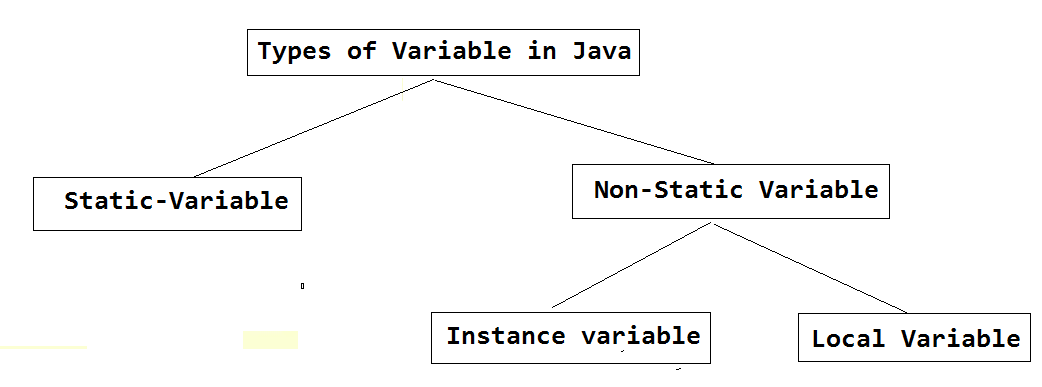
b = 25; // ------> direct assigned variable

b = a; // ------> assigned value in term of variable

b = a+15; // ------> assigned value as term of expression

**Types of Variable in Java:**

**Version-2:**



**Static or Class variables**

* These variables are declared inside the class and outside methods, with the modifier “static”.
* These are also called class variables, because all the instances of a class share the same variable (all instances of a class point to the same memory location of the static variable).
* If an instance modifies the static variable, the change is reflected in all other instances of the class. Hence the scope of these type of variables is the class itself.
* If it is desired that the value of a static variable should be kept constant, making instances incapable of changing the value, “final” modifier should be used during variable declaration.
* Any attempt to change the value of final static variable, results in compile error.

**Non-static or Instance variables**

* These variables are declared inside the class and outside methods, without the modifier “static”.
* These are also called instance variables, because the variable value could differ from one instance to another instance of the same class. Hence, the variable value is unique to an instance of class, not the class itself, unlike class variables.
* Any change in the value of non-static variable, the change is confined only to that instance in which the change happened. It doesn’t affect other instances of the class. Hence the scope of these type of variables is the entire instance of a class.
* As these type of variable values are unique to an instance, they define the state of the object/instance of the class.

**Version-2:**

**Division 1:** Based on the type of value represented by a variable all variables are divided into 2 types. They are:

1. Primitive variables

2. Reference variables

**Primitive variables:**

Primitive variables can be used to represent primitive values.

Example: int x=10;

**Reference variables:**

Reference variables can be used to refer objects.

Example: Student s=new Student();

**Diagram:**



**Division 2 :** Based on the behaviour and position of declaration all variables are divided into the following 3 types.

1. Instance variables

2. Static variables

3. Local variables

**Instance variables:**

* If the value of a variable is varied from object to object such type of variables are called instance variables.
* Instance variables should be declared with in the class directly but outside of any method or block or constructor.

**Ex:**

public class InstanceEmployee1 {

//instance variable

int e\_no;

String e\_name;

double e\_salary;

//constructor

public InstanceEmployee1(int id,String n,double s){

e\_no=i;

e\_name=n;

e\_salary=s;

}

//method

public void display(){

System.out.println(e\_no+"---"+e\_name+"---"+e\_salary);

}

public static void main(String[] args) {

InstanceEmployee1 emp1=new InstanceEmployee1();

System.out.println(emp1.e\_no+"---"+emp1.e\_name+"---"+emp1.e\_salary);

InstanceEmployee1 emp2=new InstanceEmployee1();

System.out.println(emp2.e\_no+"---"+emp2.e\_name+"---"+emp2.e\_salary);

}

}

**For every object a separate copy of instance variables will be created.**

**Ex:**

public class InstanceEmployee1 {

//instance variable

int e\_no;

String e\_name;

double e\_salary;

//constructor

public static void main(String[] args) {

InstanceEmployee1 emp1=new InstanceEmployee1();

emp1.e\_no=101;

emp1.e\_name="veeru";

emp1.e\_salary=40000;

System.out.println(emp1.e\_no+"---"+emp1.e\_name+"---"+emp1.e\_salary);

InstanceEmployee1 emp2=new InstanceEmployee1();

emp2.e\_no=102;

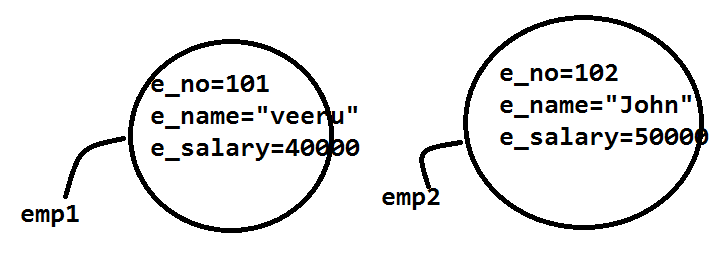
emp2.e\_name="John";

emp2.e\_salary=50000;

System.out.println(emp2.e\_no+"---"+emp2.e\_name+"---"+emp2.e\_salary);

}

}



* Instance variables will be created at the time of object creation (with the use of the keyword 'new') and destroyed at the time of object destruction hence the scope of instance variables is exactly same as scope of objects.
* Instance variables will be stored on the heap as the part of object.
* When a space is allocated for an object in the heap, a slot for each instance variable value is created.
* Instance variables can be accessed directly from Instance area. But cannot be accessed directly from static area.

**Ex:**

class Employee {

//instance variable

int id=101;

//static area

public static void main(String[] args) {

//System.out.println(num);

//C.E: non-static variable num cannot be referenced from a static context

}

}

* But by using object reference we can access instance variables from static area.

**Ex:**

class Employee {

//instance variable

int id=101;

//static area

public static void main(String[] args) {

//System.out.println(num);

//C.E: non-static variable num cannot be referenced from a static context

//creating Object

Employee emp1=new Employee();

//access throgh ref variable

System.out.println(emp1.id);

}

}

**Note:** Access modifiers can be given for instance variables.

* The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers

**Ex:**

public class InstanceEmployee {

// this instance variable is visible for any child class.

public String name;

// salary variable is visible in Employee class only.

private double salary;

//The name variable is assigned in the constructor.

public InstanceEmployee(String name) {

this.name = name;

}

// The salary variable is assigned a value.

public void setSalary(double salary) {

this.salary = salary;

}

// This method prints the employee details.

public void printEmp(){

System.*out*.println("Employee name: "+name);

System.*out*.println("Employee salary: "+salary);

}

public static void main(String[] args) {

InstanceEmployee emp1=new InstanceEmployee("Veeru");

emp1.setSalary(100000);

emp1.printEmp();

}

}

**Note:** For the instance variables it is not required to perform initialization JVM will always provide default values.

**Ex:**

public class InstanceEmployee1 {

//instance variable

int e\_no;

String e\_name;

double e\_salary;

public static void main(String[] args) {

InstanceEmployee1 emp1=new InstanceEmployee1();

System.*out*.println(emp1.e\_no+"---"+emp1.e\_name+"---"+emp1.e\_salary);

InstanceEmployee1 emp2=new InstanceEmployee1();

System.*out*.println(emp2.e\_no+"---"+emp2.e\_name+"---"+emp2.e\_salary);

}

}

**Output:**

0 null 0.0

0 null 0.0

**Static variables:**

* If the value of a variable is not varied from object to object such type of variables is not recommended to declare as instance variables. We have to declare such type of variables at class level by using static modifier.

**Ex: Without using static variable**

public class StaticEmployee1 {

//instance variable

int e\_no;

String e\_name;

double e\_salary;

String company\_name="SynchroServe";

public static void main(String[] args) {

InstanceEmployee1 emp1=new InstanceEmployee1();

emp1.e\_no=101;

emp1.e\_name="veeru";

emp1.e\_salary=40000;

System.out.println(emp1.e\_no+"---"+emp1.e\_name+"---"+emp1.e\_salary+"--"+emp1.company\_name);// 101---veeru---40000.0--SynchroServe

InstanceEmployee1 emp2=new InstanceEmployee1();

emp2.e\_no=102;

emp2.e\_name="John";

emp2.e\_salary=50000;

System.out.println(emp2.e\_no+"---"+emp2.e\_name+"---"+emp2.e\_salary+"--"+emp2.company\_name);// 102---John---50000.0--SynchroServe

}

}

**Ex: with static variable:**

public class StaticEmployee1 {

//instance variable

int e\_no;

String e\_name;

double e\_salary;

static String company\_name="SynchroServe";

public static void main(String[] args) {

InstanceEmployee1 emp1=new InstanceEmployee1();

emp1.e\_no=101;

emp1.e\_name="veeru";

emp1.e\_salary=40000;

System.out.println(emp1.e\_no+"---"+emp1.e\_name+"---"+emp1.e\_salary+"--"+emp1.company\_name); // 101---veeru---40000.0--SynchroServe

InstanceEmployee1 emp2=new InstanceEmployee1();

emp2.e\_no=102;

emp2.e\_name="John";

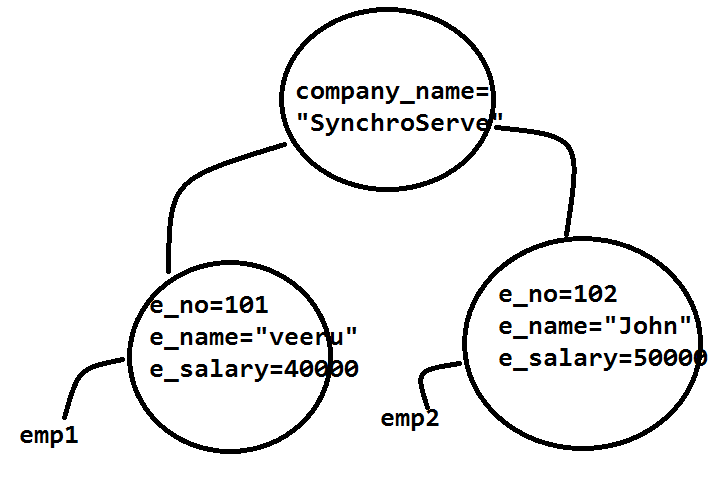
emp2.e\_salary=50000;

System.out.println(emp2.e\_no+"---"+emp2.e\_name+"---"+emp2.e\_salary+"--"+emp2.company\_name); // 102---John---50000.0--SynchroServe

}

}

* In the case of instance variables for every object a separate copy will be created but in the case of static variables for entire class only one copy will be created and shared by every object of that class.



* Static variables will be crated at the time of class loading and destroyed at the time of class unloading hence the scope of the static variable is exactly same as the scope of the .class file.
* Static variables will be stored in method area. Static variables should be declared with in the class directly but outside of any method or block or constructor
* Static variables can be accessed from both instance and static areas directly.

**Ex:**

public class Test1 {

static int num=10;

//instace area

public void m1(){

System.out.println(num);

}

public static void main(String[] args) {

//directly

System.out.println(num);//10

Test1 t1=new Test1();

t1.m1();

}

}

* We can access static variables either by class name or by object reference but usage of class name is recommended.
* But within the same class it is not required to use class name we can access directly

**Ex:**

public class Test1 {

static int num=10;

public static void main(String[] args) {

//directly

System.out.println(num);//10

//with class name

System.out.println(Test1.num);//10

Test1 t1=new Test1();

//with obj ref

System.out.println(t1.num);//10

}

}

**Note:** For the static variables it is not required to perform initialization explicitly, JVM will always provide default values.

**Ex:**

public class Test1 {

static int num;

static String name;

static boolean a;

public static void main(String[] args) {

System.out.println(num);//0

System.out.println(name);//null

System.out.println(a);//false

}

}

**Ex-2:**

public class Test1 {

static int a=10;

int b=20;

public static void main(String[] args) {

Test1 t1=new Test1();

System.out.println(t1.a);//10

System.out.println(t1.b);//20

t1.a=111;

t1.b=222;

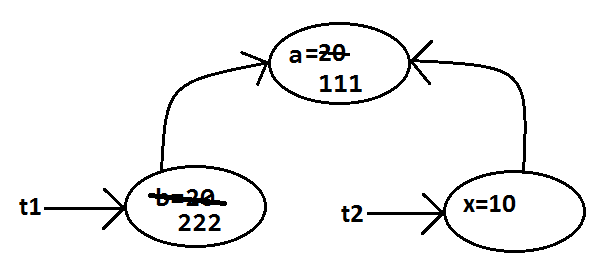
Test1 t2=new Test1();

System.out.println(t2.a);//111

System.out.println(t2.b);//20

}

}



**Note:**

Static variables also known as class level variables or fields.

**Local variables:**

* Some times to meet temporary requirements of the programmer we can declarevariables inside a method or block or constructors such type of variables are called local variables or automatic variables or temporary variables or stack variables.
* Local variables will be stored inside stack.

**Ex:**

class Test {

public static void main(String[] args) {

int i=10;

for (int j=0;j<=10 ;j++ ){

//here j is the local variable to for loop

System.out.println("value of J: "+j);

}

System.out.println("value of J: "+j);// error: cannot find symbol

}

}

**Ex:**

class Test {

public void m1(){

int i=10;

System.out.println("value of J: "+j);//error: cannot find symbol

}

public void m2(){

int j=10;

System.out.println("value of J: "+i);/error: cannot find symbol

}

public static void main(String[] args) {

Test t1=new Test();

t1.m1();

t1.m2();

System.out.println("value of J: "+j);// error: cannot find symbol

}

}

* The local variables will be created as part of the block execution in which it is declared and destroyed once that block execution completes. Hence the scope of the local variables is exactly same as scope of the block in which we declared.

**Ex:**

public class Student {

public void getAge() {

//local variable age

int age = 0;

age = age + 5;

System.out.println("Student age is : " + age); //5

}

public static void main(String args[]) {

Student s1 = new Student();

s1.getAge();

}

}

**Ex:**

**Ex:**

public class Student {

public void getAge() {

//local variable age

int age = 0;

age = age + 5;

System.out.println("Student age is : " + age);

}

public static void main(String args[]) {

System.out.println("Student age is : " + age);

}

}

* The local variables will be stored on the stack.
* For the local variables JVM won't provide any default values compulsory we should perform initialization explicitly before using that variable.

**Ex:**

class Test {

public static void main(String[] args) {

int j;

System.out.println("value of J: "+j);//error: variable j might not have been initialized

}

}

**Here not using local variable**

class Test {

public static void main(String[] args)

{

int j;

System.out.println("here we not using j");

}

}

**Ex:**

class Test {

public static void main(String[] args) {

int j;

System.out.println("value of J: "+j);//error: variable j might not have been initialized

}

}

* It is never recommended to perform initialization for the local variables inside logical blocks because there is no guarantee of executing that block always at runtime.

**Ex:**

class Test {

public static void main(String[] args) {

int i;

if (args.length>0){

System.out.println("value of J: "+i);//error: variable i might not have been initialized

}

}

}

**Ex:**

class Test{

public static void main(String[] args) {

int i;

if (args.length>0){

i=10;

System.out.println("value of J: "+i);//error: variable i might not have been initialized

}

else{

i=20;

System.out.println("value of J: "+i);

}

}

}

**Output:**

F:\>javac Test.java

F:\>java Test 10

value of J: 10

F:\>java Test

value of J: 20

* It is highly recommended to perform initialization for the local variables at the time of declaration at least with default values.

**Ex:**

class Test{

public static void main(String[] args) {

int i=0;

System.out.println("value of i: "+i);

}

}

**Note:** The only applicable modifier for local variables is final. If we are using any other modifier we will get compile time error.

**Example:**

class Test {

public static void main(String[] args) {

public int x=10; //(invalid)

private int x=10; //(invalid)

protected int x=10; //(invalid) C.E: illegal start of expression

static int x=10; //(invalid)

volatile int x=10; //(invalid)

transient int x=10; //(invalid)

final int x=10;//(valid)

}

}

**Instance variable Vs Static variable**

* Each object will have its own copy of instance variable whereas We can only have one copy of a static variable per class irrespective of how many objects we create.
* Changes made in an instance variable using one object will not be reflected in other objects as each object has its own copy of instance variable. In case of static, changes will be reflected in other objects as static variables are common to all object of a class.
* We can access instance variables through object references and Static Variables can be accessed directly using class name.

**Conclusions:**

1. For the static and instance variables it is not required to perform initialization explicitly JVM will provide default values. But for the local variables JVM won't provide any default values compulsory we should perform initialization explicitly before using that variable.

2. For every object a separate copy of instance variable will be created whereas for entire class a single copy of static variable will be created. For every Thread a separate copy of local variable will be created.

3. Instance and static variables can be accessed by multiple Threads simultaneously and hence these are not Thread safe but local variables can be accessed by only one Thread at a time and hence local variables are Thread safe.

4. If we are not declaring any modifier explicitly then it means default modifier but this rule is applicable only for static and instance variables but not local variable.

**UnInitialized arrays:**

**Local level:**

**Ex:**

class Test {

int[] a;

public static void main(String[] args) {

Test t1=new Test();

System.out.println(t1.a);//null

System.out.println(t1.a[0]);//R.E:NullPointerException

}

}

**Ex: with default values:**

class Test {

int[] a=new int[5];

public static void main(String[] args) {

Test t1=new Test();

System.out.println(t1.a);//null

System.out.println(t1.a[0]);//R.E:NullPointerException

}

}

**Ex: Static level:**

class Test {

static int[] a;

public static void main(String[] args) {

Test t1=new Test();

System.out.println(t1.a);//null

System.out.println(t1.a[0]);//R.E:NullPointerException

}

}

**Ex:**

class Test {

static int[] a=new int[5];

public static void main(String[] args) {

Test t1=new Test();

System.out.println(t1.a);//null

System.out.println(t1.a[0]);//R.E:NullPointerException

}

}

**Local level:**

**Ex:**

class Test {

public static void main(String[] args) {

int[] a;

System.out.println(a); //C.E: variable a might not have been initialized

System.out.println(a[0]);

}

}

**Ex:**

class Test {

public static void main(String[] args) {

int[] a=new int[3];

System.out.println(a);//[I@3e25a5

System.out.println(a[0]);//0

}

}

Once we created an array every element is always initialized with default values irrespective of whether it is static or instance or local array.

**Note:**

Every variable in java should be either instance or static or local.

Every variable in java should be either primitive or reference

Hence the following are the various possible combinations for variables

