## **Types of Variables in Java**

There are **three types of variables** in Java.

1) Local variable 2) Static (or class) variable 3) Instance variable

## **Static (or class) Variable**

Static variables are also known as class variable because they are associated with the class and common for all the instances of class. For example, If I create three objects of a class and access this static variable, it would be common for all, the changes made to the variable using one of the object would reflect when you access it through other objects.

### **Example of static variable**

public class StaticVarExample {  
 public static String myClassVar="class or static variable";  
   
 public static void main(String args[]){  
 StaticVarExample obj = new StaticVarExample();  
 StaticVarExample obj2 = new StaticVarExample();  
 StaticVarExample obj3 = new StaticVarExample();  
   
 //All three will display "class or static variable"  
 System.out.println(obj.myClassVar);  
 System.out.println(obj2.myClassVar);  
 System.out.println(obj3.myClassVar);  
  
 //changing the value of static variable using obj2  
 obj2.myClassVar = "Changed Text";  
  
 //All three will display "Changed Text"  
 System.out.println(obj.myClassVar);  
 System.out.println(obj2.myClassVar);  
 System.out.println(obj3.myClassVar);  
 }  
}

**Output:**

class or static variable  
class or static variable  
class or static variable  
Changed Text  
Changed Text  
Changed Text

As you can see all three statements displayed the same output irrespective of the instance through which it is being accessed. That’s is why we can access the static variables without using the objects like this:

System.out.println(myClassVar);

Do note that only static variables can be accessed like this. This doesn’t apply for instance and local variables.

## 

## **Instance variable**

Each instance(objects) of class has its own copy of instance variable. Unlike static variable, instance variables have their own separate copy of instance variable. We have changed the instance variable value using object obj2 in the following program and when we displayed the variable using all three objects, only the obj2 value got changed, others remain unchanged. This shows that they have their own copy of instance variable.

### **Example of Instance variable**

public class InstanceVarExample {  
 String myInstanceVar="instance variable";  
   
 public static void main(String args[]){  
 InstanceVarExample obj = new InstanceVarExample();  
 InstanceVarExample obj2 = new InstanceVarExample();  
 InstanceVarExample obj3 = new InstanceVarExample();  
   
 System.out.println(obj.myInstanceVar);  
 System.out.println(obj2.myInstanceVar);  
 System.out.println(obj3.myInstanceVar);  
  
   
 obj2.myInstanceVar = "Changed Text";  
  
   
 System.out.println(obj.myInstanceVar);  
 System.out.println(obj2.myInstanceVar);  
 System.out.println(obj3.myInstanceVar);  
 }  
}

**Output:**

instance variable  
instance variable  
instance variable  
instance variable  
Changed Text  
instance variable

## 

## **Local Variable**

These variables are declared inside method of the class. Their scope is limited to the method which means that You can’t change their values and access them outside of the method.

In this example, I have declared the instance variable with the same name as local variable, this is to demonstrate the scope of local variables.

### **Example of Local variable**

public class VariableExample {  
 // instance variable  
 public String myVar="instance variable";  
   
 public void myMethod(){  
 // local variable  
 String myVar = "Inside Method";  
 System.out.println(myVar);  
 }  
 public static void main(String args[]){  
 // Creating object  
 VariableExample obj = new VariableExample();  
   
 /\* We are calling the method, that changes the   
 \* value of myVar. We are displaying myVar again after   
 \* the method call, to demonstrate that the local   
 \* variable scope is limited to the method itself.  
 \*/  
 System.out.println("Calling Method");  
 obj.myMethod();  
 System.out.println(obj.myVar);  
 }  
}

**Output:**

Calling Method  
Inside Method  
instance variable

If I hadn’t declared the instance variable and only declared the local variable inside method then the statement System.out.println(obj.myVar); would have thrown compilation error. As you cannot change and access local variables outside the method.