Variables in Java | Types of Variables -Siva Yannam

A **variable in Java** is a container that holds the value during the execution of Java program. In other words, Variable is the name of the memory location reserved for storing value.
Each variable in Java has a specific type that determines size of the memory. The size of the memory reserved depends on the data type.

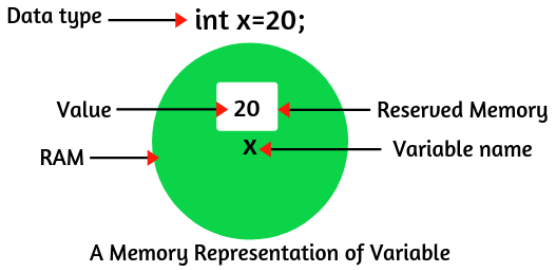
There are two types of data types in Java:

- Primitive Data type
- Non-primitive Data type

A variable with a primitive type holds the value of primitive type only. A variable with reference type (non-primitive) holds either a null reference or a reference to an object of the same type as the variable.

For example, if we write int x = 20, the variable name is x which stores the value 20 in a location of memory, where int is a primitive data type that represents that this variable can store only integer values.

In other words, it can only take integer values. The value stored in any variable can be changed during program execution.



A Memory Representation of Variable

Variable Declaration in Java

In Java, all the variables must be declared to the java compiler before they can be used in the program. We can declare a variable in Java by using two types of syntax:

Syntax:

```
1. data_type  variable_name;

2. data_type  variable_name = value;
```

You must end the declaration statement with a semicolon.

For example:

int a; where int is a data type and a is variable.
int b = 20; where int is a data type and b is a variable. Here, we can say that b is a type of int having value as twenty.

When we define a variable with some initial value, it is called **initialization**. A variable must be given a value after declaration but before it has been used in the expression.
To declare more than one variable of the same type, we can use a comma-separated list.

Following are the valid examples of the variables declaration and initialization in Java:

```
int x, y, z; //  Here, we declare three variables x, y, and z having data type int.

int x = 10, y = 20; // Example of Initialization.

byte a = 30; // Initializes a byte type variable a.

double pi = 3.14; // declares and assigns a value of pi.
```

Variable Initialization in Java

The value of variables can be assigned in two ways:

1. Static: When the class is loaded into the memory, the memory is determined for the variables. Such variables are known as static variables or class variables in java.
For example:

```
static String name = "Kiran"; // Here, name is variable having string value and variable name is declared with the static keyword.
```

2. Dynamic: In dynamic initialization, you can declare variables anywhere in the program because when the statement is executed, the memory is assigned to them.
For example:

```
char ch = 'B'; // Declaring and Initializing character variables.

int number = 100; // Declaring and Initializing integer variables.

int time = 40, distance =  50;
```

Naming Convention for the declaring variables in Java

There are some important points that need to keep in mind during the declaration of java variables. They are as follows:

1. As per Java coding standard, the variable name should start with a lower case letter.

For example:

int age;

If you have lengthy variables such as more than one words then you can declare the first word small and second word with the capital letter like this:

int smallNumber;
String collegeName;

2. The variable name should not contain a blank space.

int num ber = 100; is invalid because there is a blank space between num and ber.

3. The variable name can begin with a special character such as $ and _.

For examples:

String $name; // valid.
String _nSchool; // valid.
int @num; // invalid.

4. The first character must be a letter. For example, int 2num; // invalid.

5. Java keywords should not be used as a variable name.

6. The variable names are case sensitive in Java.

Types of Variables in Java

There are three types of variables in java. They are as:

1. **Local variables**
2. **Instance variables**
3. **Class/Static variables**

Local Variables in Java

1. A variable that is declared and used inside the body of methods, constructors, or blocks is called **local variable in java**. It is called so because local variables are not available for use from outside.

2. Local variables must be assigned a value at the time of creating. If you use a local variable without initializing a value, you will get a compile-time error like "variable x not have been initialized".
For example:

```
public void mySchool()

{

 // Declaration of local variables.

     String schoolName;    // Compilation error due to not initializing of value.

     System.out.println("Name of School: " +schoolName);

}
```

3. No access modifiers can be used with local variables.
4. The local variables are visible only within the declared constructor, method, or block.
5. A local variable is not equivalent to an instance variable.
6. A local variable cannot be static.

Let's take an example program based on important points of local variables.

Program source code 1:

```
package localVariables;

 public class Student
```

```
 {

// Declaration of constructor.

   Student()

   {

     String nCollege = "PIET"; // Declaration and initialization of local variable.

     System.out.println("Name of college: " +nCollege); // We can access local variable
inside the constructor.

    }

// Declaration of instance method.

  void subMarks()

  {

// Declaration and initialization of local variables.

   int cMarks = 90;

   int pMarks = 85;

   int mMarks = 99;

   int totalMarks = cMarks + pMarks + mMarks;

   System.out.println("Total marks in PCM: " +totalMarks);

  }

public static void main(String[] args)

{
```

```
// Create an object of class.

    Student s = new Student();

// System.out.println("Name of college: " +nCollege); // Compile-time error because local variable cannot be accessed from the outside.



    s.subMarks(); // Calling instance method.

 // System.out.println("Total marks in PCM: " + totalMarks); // Compile-time error.

   }

}
```

Output:

```
        Name of college: PIET

        Total marks in PCM: 274
```

Let's take another program where we will declare access modifiers with local variable and see what we get?

Program source code 2:

```
package localVariables;

public class Test

{

 void m1() // Instance method.

 {

  public int x = 20; // Compile-time error because public is access modifier and we cannot declare access modifiers with local variables.
```

```
 }

public static void main(String[] args)

{

  Test t = new Test(); // Creating object of class Test.

   t.m1(); // Calling m1 method.

  }

}

Output:

          Unresolved compilation problem: Illegal modifier for parameter x; only final is
permitted
```

As you can see in the above example program, only final keyword is permitted with local variables in the java program. You will learn about final keyword in the further tutorial.

Memory Allocation of Local Variables in Java

When the method starts, the memory for the local variable is allocated. When the method is completed, the memory of the local variable is released.

For example:

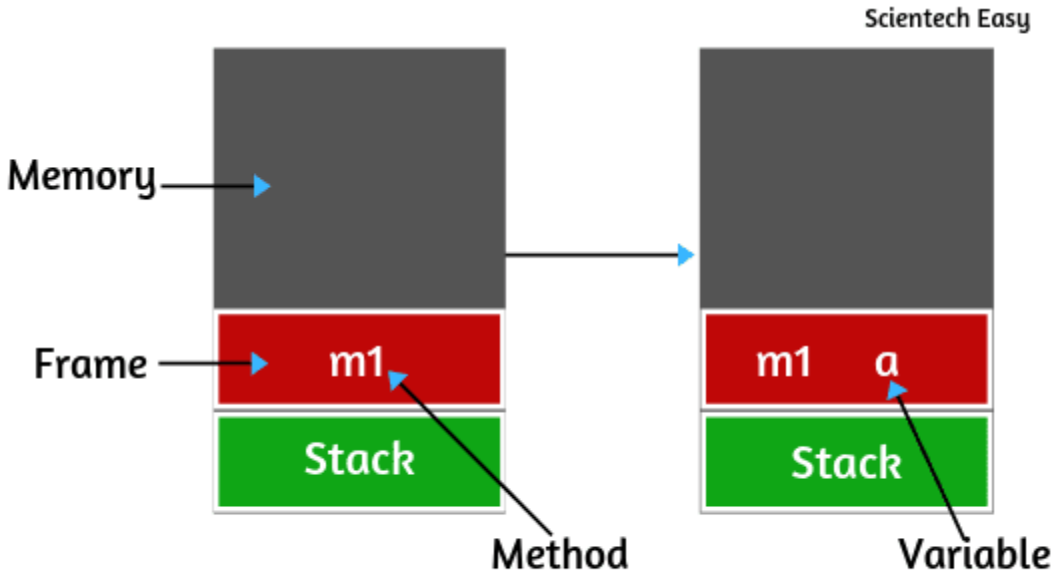
```
void m1() // Memory allocated when the method starts.

 {
```

```
  // Declaration of local variables.

     int a = 30;

     int b = 40; // Logic here.

} // Memory released when the method is completed.
```

Stored Memory of Local variables

Local variables are stored in stack memory. Consider the preceding example program. When the main method calls m1() method, a frame is created in the stack memory for the method m1().

After creating a frame for method m1, a variable 'a' in method m1 is created in the frame of the stack memory as shown in the below figure.



A stored memory representation of Local variable.

Instance Variables in Java

1. A variable that is declared inside the class but outside the body of the method, constructor, or any block is called **instance variable in java**. They are available for the entire class methods, constructors, and blocks. It is also called non-static variable because it is not declared as static.

2. Instance variables are created when an object is created using the keyword 'new' and destroyed when the object is destroyed.

3. We can also use access modifiers with instance variables. If we do not specify any modifiers, the default access modifiers will be used which can be accessed in the same package only.

4. It is not necessary to initialize the instance variable.

Let's create a program where we will declare instance variables and will see how to access instance variables from the static and non-static area.

Program source code 3:

```
package instanceVariables;

 public class Test

 {

// Declare instance variables inside the class, not inside the method, constructor, or block.

    int a = 30;

    int b = 40;

// This method is called Static method.

   public static void main(String[] args) // main method.

   {
```

```
// This area is called static area. So, you can access instance variables by using an object reference variable.

// Creating an object of the class Test.

    Test t = new Test();

    System.out.println(t.a); // Accessing instance variable using object reference variable 't'.

    System.out.println(t.b);

    }

// Declaration of instance method.

   void m1() // This method is called user-defined method.

   {

// This area is called instance Area. So, we can access directly instance variables without creating any object.

     System.out.println(a); //Direct access.

     System.out.println(b); //Direct access.

     }

 }
```

Output:

30

40

Memory allocation of Instance Variables in Java

1. Instance variables are variables of the object which are commonly known as field or object variables. When an object is created for a class, the memory is allocated for the instance variables and destroyed when the object is destroyed.

2. Each instance (object) of a class has its own copy of each variable. In other words, instance variables have their own separate copy of instance variable. If one object changes the value of a variable, it does not affect the value of other instance variables.

Let's make a program where we will declare an instance variable and create two objects of the class. Here, we will understand that if one object will change the value of a variable, does it affect the value of another instance variable?

Program source code 4:

```
package instanceVariables;

 public class Marks

 {

// Declare the instance variable inside the class.

     int phyMarks = 80;

public static void main(String[] args)

 {

// Create the two objects of the class 'Marks'.
```

```
    Marks m1 = new Marks();

    Marks m2 = new Marks();



// Call the variables using object m1 and m2.

    int pMarks1 = m1.phyMarks;

    int pMarks2 = m2.phyMarks;

    System.out.println("Marks in Physics: " +pMarks1);

    System.out.println("Marks in Physics: " +pMarks2);



/* If we change the value of instance variable using object reference m2, the value of
object m1 variable will not change. Only the value of instance variable calling by using
object m2 will change. This shows that they have their own copy of instance variable.*/



    m2.phyMarks=90;

    System.out.println("After changing the value of instance variable using object m2 ");

    System.out.println("Marks in Physics: " +m1.phyMarks);

    System.out.println("Marks in Physics: " +m2.phyMarks);

  }

}
```

Output:

```
        Marks in Physics: 80
```

```
Marks in Physics: 80

After changing the value of instance variable using object m2.

Marks in Physics: 80

Marks in Physics: 90
```

Thus, you can change the value of instance variable by using an object if need.

Stored memory of Instance variables in Java

When we make any program, each program has its own memory known as **Run time Data Area** which is allocated by JVM (Java Virtual Machine). JVM uses two memory "Stack" and "Heap" memory for storing data-type. When we create an object using the new keyword, their related object variables and properties are stored in memory.

All instance variables are stored in PermGen space (Permanent Generation space) of heap memory. If the variables are primitive type then variable and its value both are stored as a name-value pair in the permgen.

But if the variable is user-defined (object) then its reference is stored in PermGen but actually, it is stored in Young/old generation of heap memory.

Static Variables in Java

1. A variable which is declared with a static keyword is called **static variable in java**. A static variable is also called class variable because it is associated with the class.
2. Static variables are always declared inside the class but outside of any methods, constructors, or blocks.

Let's take an example program based on static variables.

Program source code 5:

```
package staticVariables;

 public class Test

 {

// Declaration of static variables.

    static int a = 400;

    static int b = 500;

// Static method or main method.

    public static void main(String[] args)

    {

// Static area. So, we can call static variables or static methods by using the class name.

     System.out.println(Test.a); // Test is the name of class.

     System.out.println(Test.b);

      Test t = new Test(); // Creating the object of class Test.

       t.m1;

    }
```

```
// Instance method.

    void m1()

    {

// Instance area.

     System.out.println(Test.a);

     System.out.println(Test.b);

    }

}

Output:

          400

          500

          400

          500
```

In the above program, we must call m1() method by creating the object of the class in static region. Otherwise, it will not print output on the console.

3. Static variable will get the memory only once. If anyone changes the value of the static variable using the class name, it will replace the previous value and display the changed value. This is because it is constant for every object created.

Let's see an example program based on this concept.

Program source code 6:

```
package staticVariables;
```

```
class College

 {

// Static variable.

  static String collegeName = "PIET";

// static method.

  public static void main(String[] args)

  {

    System.out.println(College.collegeName);

// Suppose anyone changes the value of a static variable using the class name. In this case it will display changed value.

   College.collegeName = "RSVM";

   System.out.println(College.collegeName);

   }

}
```

Output:

PIET

RSVM

Memory Allocation of Static variables

Memory allocation for static variables happens only once when the dot class is loaded into the memory and it is destroyed when dot class unloaded into the memory.

Stored memory of Static variables

All the static variables are stored in PermGen space of heap memory.