Java Variables

In Java, variables are containers that store data in memory. Understanding variables plays a very important role as it defines how data is stored, accessed, and manipulated.

Key Components of Variables in Java:

A variable in Java has three components, which are listed below:

- **Data Type:** Defines the kind of data stored (e.g., int, String, float).
- **Variable Name:** A unique identifier following Java naming rules.
- **Value:** The actual data assigned to the variable.

Note: There are three types of variables in Java: **Local, Instance** and **Static**.

Example: The below example demonstrates the **variable declaration in Java**

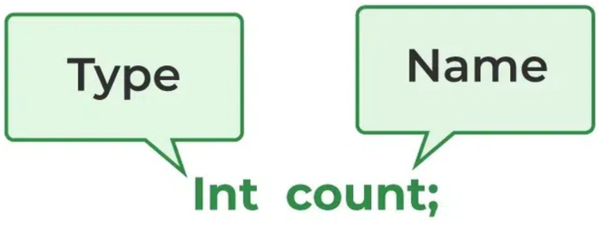
```java
// Demonstarting how to declare and use a variable in Java

class Gla {
   public static void main(String[] args) {
      // Declaring and initializing variables

      // Integer variable
      int age = 25;

      // String variable
      String name = "GlaforGla";

      // Double variable
      double salary = 50000.50;

      // Displaying the values of variables
      System.out.println("Age: " + age);
      System.out.println("Name: " + name);
      System.out.println("Salary: " + salary);
   }
}
```

Output

```
Age: 25

Name: GlaforGla

Salary: 50000.5
```

How to Declare Java Variables?

The image below demonstrates how we can declare a variable in Java:



Variable Declaration
From the image, it can be easily perceived that while declaring a variable, we need to take care of two things that are:

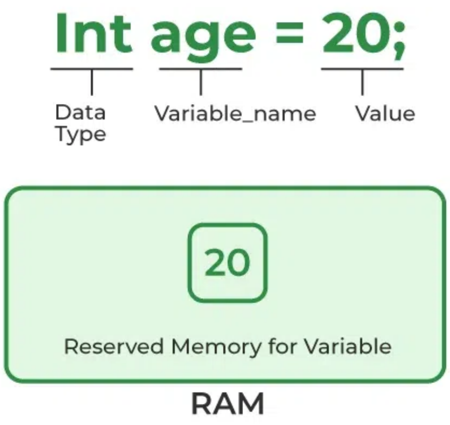
1. **data type**: In Java, a data type define the type of data that a variable can hold.
2. **variable name:** Must follow Java naming conventions (e.g., camelCase).

In this way, a name can only be given to a memory location. It can be assigned values in two ways:

- Variable Initialization
- Assigning value by taking input

How to Initialize Java Variables?

It can be perceived with the help of 3 components explained above:



Variable Initialization

Example: Here, we are **initalizing variables** of different types like **float, int and char.**

```
// Demonstrating how to intialize variables
// of different types in Java
```

```
class Gla{
  public static void main(String[] args) {
    // Declaring and initializing variables

    // Initializing float variable
    float si = 5.5f;

    // Initializing integer variables
    int t = 10;
    int s = 20;

    // Initializing character variable
    char var = 'h';

    // Displaying the values of the variables
    System.out.println("Simple Interest: " + si);
    System.out.println("Speed: " + s);
    System.out.println("Time: " + t);
    System.out.println("Character: " + var);
  }
}
```

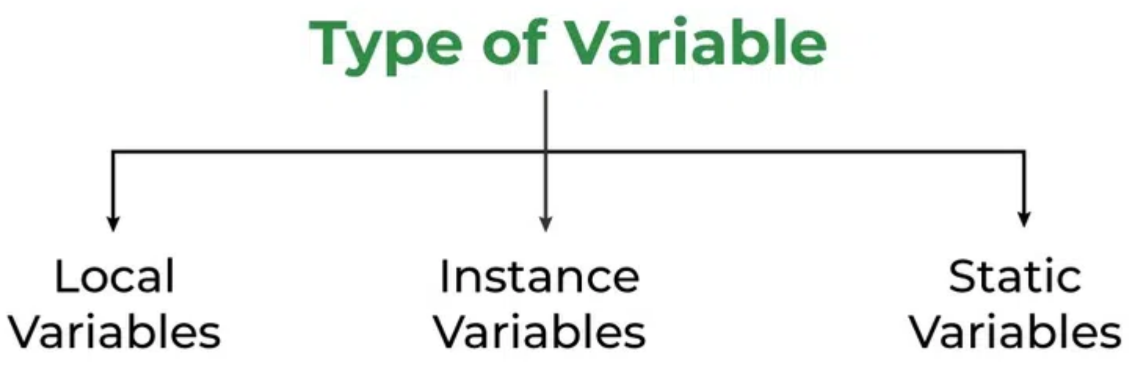
Output

```
Simple Interest: 5.5

Speed: 20

Time: 10

Character: h
```

Types of Java Variables

Now let us discuss different types of variables which are listed as follows:

- Local Variables
- Instance Variables
- Static Variables



Type of Variable

Let us discuss the traits of every type of variable listed here in detail.

1. Local Variables

A variable defined within a block or method or constructor is called a local variable.

- The Local variable is created at the time of declaration and destroyed when the function completed its execution.
- The scope of local variables exists only within the block in which they are declared.
- We first need to initialize a local variable before using it within its scope.

Example: This example show how a **local variable** is declared and used inside the main method and it can not be used outside of it.

```java
// Java Program to show the use of local variables
import java.io.*;

class Gla {
   public static void main(String[] args)
   {
      // Declared a Local Variable
      int var = 10;

      // This variable is local to this main method only
      System.out.println("Local Variable: " + var);
   }
}
```

Output

```
Local Variable: 10
```

Example: This example demonstrates that local variables are only accessible within the block in which they are declared

```java
// Java Program to show the use of
// Local Variables
import java.io.*;

public class Gla {

  public static void main(String[] args)
  {
    // x is a local variable
    int x = 10;

    // message is also a local
    // variable
    String message = "Hello, world!";

    System.out.println("x = " + x);
    System.out.println("message = " + message);

    if (x > 5) {
      // result is a
      // local variable
      String result = "x is greater than 5";
      System.out.println(result);
    }

    // Uncommenting the line below will result in a
    // compile-time error System.out.println(result);
    for (int i = 0; i < 3; i++) {
      String loopMessage
        = "Iteration "
         + i; // loopMessage is a local variable
      System.out.println(loopMessage);
    }

    // Uncommenting the line below will result in a
    // compile-time error
    // System.out.println(loopMessage);
  }
}
```

Output

```
x = 10

message = Hello, world!

x is greater than 5
```

```
Iteration 0
Iteration 1
Iteration 2
```

2. Instance Variables

Instance variables are known as non-static variables and are declared in a class outside of any method, constructor, or block.

- Instance variables are created when an object of the class is created and destroyed when the object is destroyed.
- Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier, then the default access specifier will be used.
- Initialization of an instance variable is not mandatory. Its default value is dependent on the data type of variable. For *String* it is *null*, for *float* it is *0.0f*, for *int* it is *0*, for Wrapper classes like *Integer* it is *null, etc.*
- Scope of instance variables are throughout the class except the static contexts.
- Instance variables can be accessed only by creating objects.
- We initialize instance variables using constructors while creating an object. We can also use instance blocks to initialize the instance variables.

Example: This example demonstrates the use of instance variables, which are declared within a class and initialized via a constructor, with default values for uninitialized primitive types.

```
// Java Program to show the use of
// Instance Variables
import java.io.*;

class Instancevar {

  // Declared Instance Variable
  public String Gla;
  public int i;
  public Integer I;
  public Instancevar ()
  {
    // Default Constructor
    // initializing Instance Variable
    this.Gla = "Sweta Dash";
  }
  // Main Method
  public static void main(String[] args)
  {
    // Object Creation
    Instancevar name = new Instancevar ();

    // Displaying O/P
    System.out.println("Gla name is: " + name.Gla);
    System.out.println("Default value for int is "+ name.i);

    // toString() called internally
    System.out.println("Default value for Integer is: "+ name.I);
  }
}
```

Output

```
Gla name is: Sweta Dash

Default value for int is 0

Default value for Integer is: null
```

3. Static Variables

Static variables are also known as class variables.

- These variables are declared similarly to instance variables. The difference is that static variables are declared using the static keyword within a class outside of any method, constructor, or block.
- Unlike instance variables, we can only have one copy of a static variable per class, irrespective of how many objects we create.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.
- Initialization of a static variable is not mandatory. Its default value is dependent on the data type of variable. For *String* it is *null*, for *float* it is *0.0f*, for *int* it is *0*, for *Wrapper classes* like *Integer* it is *null*, etc.
- If we access a static variable like an instance variable (through an object), the compiler will show a warning message, which won't halt the program. The compiler will replace the object name with the class name automatically.
- If we access a static variable without the class name, the compiler will automatically append the class name. But for

accessing the static variable of a different class, we must mention the class name as 2 different classes might have a static variable with the same name.
- Static variables cannot be declared locally inside an instance method.
- Static blocks can be used to initialize static variables.

Example: This example demonstrates the use of static variables, which belong to the class and can be accessed without creating an object of the class.

```java
// Java Program to show the use of
// Static variables
import java.io.*;

class Staticvar {

    // Declared static variable
    public static String Gla = "Sweta Dash";

    public static void main(String[] args)
    {

        // Gla variable can be accessed without object
        // creation Displaying O/P Staticvar.Gla --> using the
        // static variable
        System.out.println("Gla Name is: " + Staticvar.Gla);

        // static int c = 0;
        // above line, when uncommented,
        // will throw an error as static variables cannot be
        // declared locally.
    }
}
```

Output

```
Gla Name is: Sweta Dash
```

Instance Variables vs Static Variables

Now let us discuss the differences between the Instance variables and the Static variables:

- Each object will have its own copy of an instance variable, whereas we can only have one copy of a static variable per class, irrespective of how many objects we create. Thus, **static variables** are good for **memory management**.

- Changes made in an instance variable using one object will not be reflected in other objects as each object has its own copy of the instance variable. In the case of a static variable, changes will be reflected in other objects as static variables are common to all objects of a class.
- We can access instance variables through object references, and static variables can be accessed directly using the class name**.**
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed. Static variables are created when the program starts and destroyed when the program stops.

Syntax: Static and instance variables

```
class Gla
{
// Static variable
static int a;
// Instance variable
int b;
}
```

Common Mistakes to Avoid

The common mistakes that can occur when working with variables in Java are listed below:

- **Using uninitialized local variables:** Accessing a local variable without initializing it leads to a compile-time error.
- **Confusing == and .equals() for Strings:** == is used to compare object references, while .equals() is used to compare the content of the strings.
- **Modifying static variables incorrectly:** Changing static variables in a multi-threaded environment can lead to thread safety issues