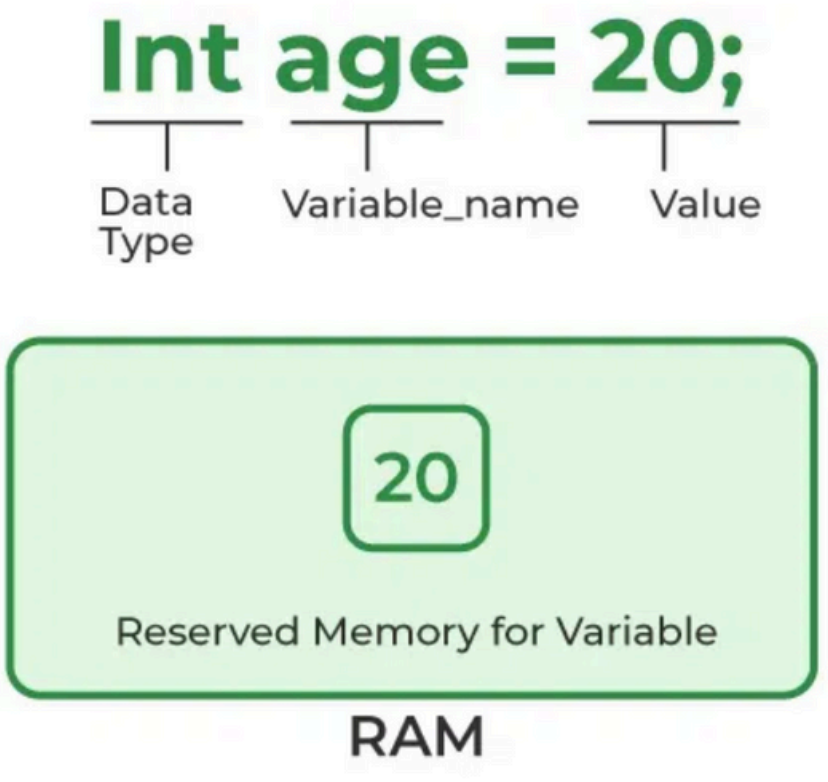
VARIABLES IN JAVA

A variable is a named memory location that holds the data value of a particular data type.

A variable in Java is a kind of container that contains the value during program execution.

Variables represents reserved storage locations, whose values can be manipulated during the execution of a program.

SYNTAX : dataType variableName ;



A variable in Java has three components,

- **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.

EXAMPLES:

```
// declaration without initialization
int x;

// declaration with initialization
int y = 10;

// multiple declarations
int a = 1, b = 2, c;
```

TYPES OF VARIABLES

DIVI SION 1 : (By type/category)

Based on the type of value represented by a variable it is divided into 2 types. They are:

1. Primitive variables
2. Reference variables

Primitive variables — hold primitive values directly:

- byte, short, int, long (integers)
- float, double (floating point)
- char (single UTF-16 code unit / character)
- boolean (true / false)

Reference variables — hold references (addresses) to objects on the heap

- e.g., String, arrays (int[]), custom classes (Student s).

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.**
- **For every object a separate copy of instance variables will be created.**
- **Instance variables will be created at the time of object creation 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.**
- **Instance variables should be declared with in the class directly but outside of and method or block or constructor.**
- **Instance variables can be accessed directly from Instance area. But cannot be accessed directly from static area.**
- **But by using object reference we can access instance variables from static area.**

```
class ClassName {
    //  Instance variable
    dataType variableName;

    // methods, constructors, etc.
}
```

```java
public class ExampleForInstanceVariable {  new *
    // Instance variable
    int number = 10;  1 usage

    void show() {  1 usage  new *
        System.out.println("Number: " + number);
    }

    public static void main(String[] args) {  new *
        ExampleForInstanceVariable obj = new ExampleForInstanceVariable(); // creating object
        obj.show();  // Output: Number: 10
    }
}
```

OUTPUT : **Number: 10**

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

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.
- 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.
- 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.

Example:

```
class Test
{
        static int i=10;
        public static void main(String[] args)
        {
                Test t=new Test();
                System.out.println(t.i);//10
                System.out.println(Test.i);//10
                System.out.println(i);//10
        }
}
```

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

Example:

```
class Test
{
        static String s;
        public static void main(String[] args)
        {
                System.out.println(s);//null
        }
}
```

Example:

```
class Test
{
        int x=10;
        static int y=20;
        public static void main(String[] args)
        {
                Test t1=new Test();
                t1.x=888;
                t1.y=999;
                Test t2=new Test();
                System.out.println(t2.x+"----"+t2.y);//10----999
        }
}
```

Local variables:

- **Some times to meet temporary requirements of the programmer we can declare variables 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.**
- **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.**
- **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.**
- **It is highly recommended to perform initialization for the local variables at the time of declaration at least with default values.**

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)
        }
}
```

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.