CHAPTER TWO

Variables and operators

Chapter Objectives

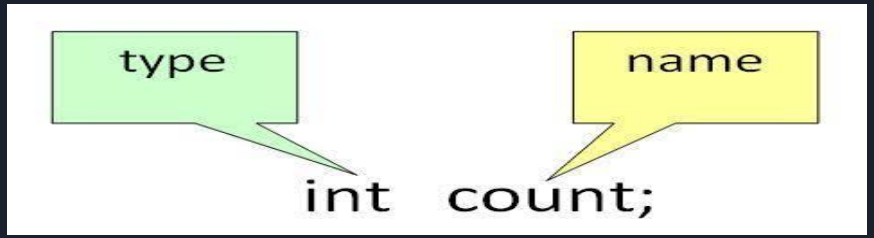
- Understanding of variables
- Types of variables
- Data types and how its used
- Primitive and non-primitive data types
- Operators used in java

Variables

- Variable in Java is a data container that saves the data values during Java program execution.
- A variable is a name given to a memory location. It is the basic unit of storage in a program.
- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location. All the operations done on the variable affect that memory location.
- In Java, all variables must be declared before use.

How to declare variables?

- We can declare variables in Java as pictorially depicted below as a visual aid.



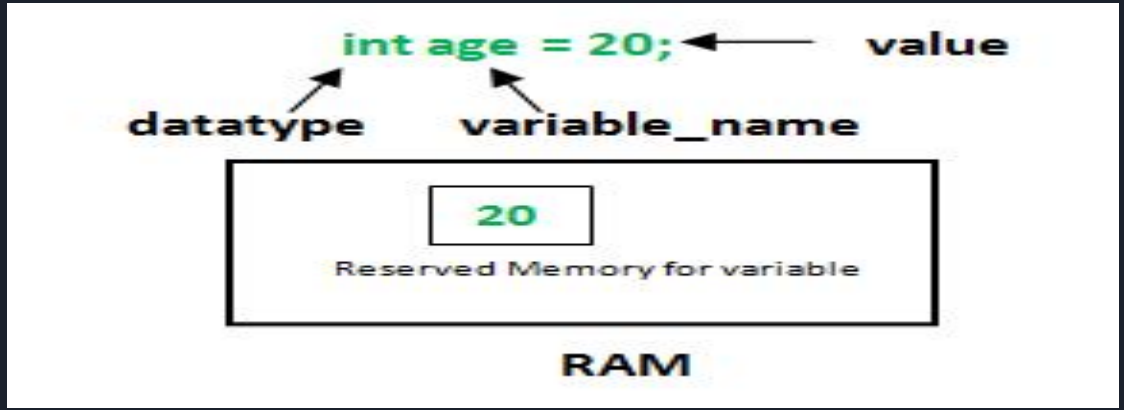
in this way, a name can only be given to a memory location. It can be assigned values in two ways:

- Variable Initialization
- Assigning value by taking input

How to initialize variables?

It can be perceived with the help of 3 components that are as follows:

- **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.



Types of Variables in Java

1. Local Variables

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

2. Instance Variables

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

3. Static Variables

   These variables are declared similarly as instance variables. The difference is that static variables are declared using the static keyword within a class outside of any method, constructor or block.

Data Types

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Primitive Data types

TYPE	DESCRIPTION	DEFAULT	SIZE	EXAMPLE LITERALS	RANGE OF VALUES
boolean	true or false	false	1 bit	true, false	true, false
byte	twos complement integer	0	8 bits	(none)	-128 to 127
char	unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\','\n',' β'	character representation of ASCII values 0 to 255
short	twos complement integer	0	16 bits	(none)	-32,768 to 32,767
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2	-2,147,483,648 to 2,147,483,647
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F	upto 7 decimal digits
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d	upto 16 decimal digits

Non-Primitive Data Types

Non-primitive data types are called reference types because they refer to objects.

The main difference between primitive and non-primitive data types are:

- Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
- Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- A primitive type has always a value, while non-primitive types can be null.
- A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
- The size of a primitive type depends on the data type, while non-primitive types have all the same size.

Type Casting

- Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

1. Widening Casting (automatically) - converting a smaller type to a larger type size
   byte -> short -> char -> int -> long -> float -> double

2. Narrowing Casting (manually) - converting a larger type to a smaller size type
   double -> float -> long -> int -> char -> short -> byte

Widening Type Casting Example

```
class Main {
  public static void main(String[] args) {
    // create int type variable
    int num = 10;
    System.out.println("The integer value: " + num);

    // convert into double type
    double data = num;
     System.out.println("The double value: " + data);
```

Narrowing type casting

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Manual casting: double to
int


    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
 }
```

Narrowing Type Casting Example

```
class Main {
  public static void main(String[] args) {
    // create double type variable
    double num = 10.99;
    System.out.println("The double value: " + num);

    // convert into int type
    int data = (int)num;
    System.out.println("The integer value: " + data);
  }
}
```

Java Operators

- Operators are symbols that perform operations on variables and values. For example, `+` is an operator used for addition, while `*` is also an operator used for multiplication
- Operators in Java can be classified into 5 types:

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operators
4. Logical Operators
5. Unary Operators
6. Bitwise Operators

Java Arithmetic Operators

- Arithmetic operators are used to perform arithmetic operations on variables and data.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Java Assignment Operator

- Assignment operators are used in Java to assign values to variables.

Operator	Example	Equivalent to
`=`	`a = b;`	`a = b;`
`+=`	`a += b;`	`a = a + b;`
`-=`	`a -= b;`	`a = a - b;`
`*=`	`a *= b;`	`a = a * b;`
`/=`	`a /= b;`	`a = a / b;`
`%=`	`a %= b;`	`a = a % b;`

Java Relational Operators

- Relational operators are used to check the relationship between two operands.

Operator	Description	Example
`==`	Is Equal To	`3 == 5` returns **false**
`!=`	Not Equal To	`3 != 5` returns **true**
`>`	Greater Than	`3 > 5` returns **false**
`<`	Less Than	`3 < 5` returns **true**
`>=`	Greater Than or Equal To	`3 >= 5` returns **false**
`<=`	Less Than or Equal To	`3 <= 5` returns **true**

Java Logical Operators

- Logical operators are used to check whether an expression is true or false. They are used in decision making.
- You can also test for true or false values with logical operators.

Operator	Example	Meaning
`&&` (Logical AND)	expression1 **&&** expression2	`true` only if both `expression1` and `expression2` are `true`
`||` (Logical OR)	expression1 **||** expression2	`true` if either `expression1` or `expression2` is `true`
`!` (Logical NOT)	**!**expression	`true` if `expression` is `false` and vice versa

Java Bitwise operators

- Bitwise operators in Java are used to perform operations on individual bits.

Operator	Description
~	Bitwise Complement
<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR