**2.2 Java Variables :-**

**Variable :** You can assign the values to the variable once it has been declared. The values of the variable can be changed anywhere in the program if the variable is accessible in that scope. In this example we have used the variable  ***intvariable*** to illustrate this.

In this section, you will learn about Java variables. A variable refers to the memory location that holds values like: numbers, texts etc. in the computer memory. A variable is a name of location where the data is stored when a program executes.

The Java contains the following types of variables:

1. **Instance Variables (Non-static fields):** In object oriented programming, objects store their individual states in the "non-static fields" that is declared without the static keyword. Each object of the class has its own set of values for these non-static variables so we can say that these are related to objects (instances of the class).Hence these variables are also known as **instance variables**. These variables take default values if not initialized.
2. **Class Variables (Static fields):** These are collectively related to a class and none of the object can claim them  its sole-proprietor . The variables defined with static keyword are shared by all objects. Here Objects do not store an individual value but they are forced to share it among themselves. These variables are declared as **"static fields"** using the **static** keyword. Always the same set of values is shared among different objects of the same class. So these variables are like **global variables** which are same for all objects of the class. These variables take default values if not initialized.
3. **Local Variables:** The variables defined in a method or block of code is called **local variables**. These variables can be accessed within a method or block of code only. These variables don't take default values if not initialized. These values are required to be initialized before using them.
4. **Parameters:** Parameters or arguments are variables used in method declarations.

**Declaring and defining variables**Before using variables you must declare the variables  name and type. See the following example for variables declaration:

**int** num;  //represents that **num** is a variable that can store value of **int** type.  
**String** name;   //represents that **name** is a variable that can store **string** value.  
**boolean** bol;  //represents that **bol** is a variable that can take **boolean** value (**true/false**);

You can assign a value to a variable at the declaration time by using an assignment operator ( **=** ).

**int** num = 1000;   // This line declares **num** as an **int** variable which holds value **"1000"**.  
**boolean** bol = true;  // This line declares **bol** as **boolean** variable which is set to the value **"true"**.

**Literals**

By literal we mean any number, text, or other information that represents a value. This means what you type is what you get. We will use literals in addition to variables in Java statement. While writing a source code as a character sequence, we can specify any value as a literal such as an integer. This character sequence will specify the syntax based on the value's type. This will give a literal as a result. For instance

int month  = 10;

In the above statement the literal is an integer value i.e 10. The literal is 10 because it directly represents the integer value.

In Java programming language there are some special type of literals that represent numbers, characters, strings and boolean values. Lets have a closer look on each of the following.

**Number Literals**Number literals is a sequence of digits and a suffix as L. To represent the type as long integer we use L as a suffix. We can specify the integers either in decimal, hexadecimal or octal format. To indicate a **decimal format** put the left most digit as nonzero. Similarly put the characters as *ox* to the left of at least one hexadecimal digit to indicate **hexadecimal format**. Also we can indicate the **octal format** by a zero digit followed by the digits 0 to 7. Lets tweak the table below.

|  |  |
| --- | --- |
| 659L | Decimal integer literal of type long integer |
| 0x4a | Hexadecimal integer literal of type integer |
| 057L | Octal integer literal of type long integer |

**Character Literals**We can specify a character literal as a single printable character in a pair of single quote characters such as 'a', '#', and '3'. You must be knowing about the ASCII character set. The ASCII character set includes 128 characters including letters, numerals, punctuations etc. There are few character literals which are not readily printable through a keyboard. The table below shows the codes that can  represent these special characters. The letter d such as in the octal, hex etc represents a number.

|  |  |
| --- | --- |
| **Escape** | **Meaning** |
| \n | New line |
| \t | Tab |
| \b | Backspace |
| \r | Carriage return |
| \f | Formfeed |
| \\ | Backslash |
| \' | Single quotation mark |
| \" | Double quotation mark |
| \d | Octal |
| \xd | Hexadecimal |
| \ud | Unicode character |

It is very interesting to know that if we want to specify a single quote, a backslash, or a nonprintable character as a character literal use an **escape sequence.** An escape sequence uses a special syntax to represents a character. The syntax begins with a single backslash character.  
Lets see the table below in which the character literals use Unicode escape sequence to represent printable and nonprintable characters both.

|  |  |
| --- | --- |
| 'u0041' | Capital letter A |
| '\u0030' | Digit 0 |
| '\u0022' | Double quote " |
| '\u003b' | Punctuation ; |
| '\u0020' | Space |
| '\u0009' | Horizontal Tab |

**Boolean Literals**The values true and false are also treated as literals in Java programming. When we assign a value to a boolean variable, we can only use these two values. Unlike C, we can't presume that the value of 1 is equivalent to true and 0 is equivalent to false in Java. We have to use the values true and false to represent a Boolean value. Like   
boolean chosen = true;  
Remember that the **literal true** is not represented by the quotation marks around it. The Java compiler will take it as a string of characters, if its in quotation marks.  
  
**Floating-point literals**Floating-point numbers are like real numbers in mathematics, for example, 4.13179, -0.000001. Java has two kinds of floating-point numbers: float and double. The default type when you write a floating-point literal is double.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Size | | Range | Precision |
| name | bytes | bits | approximate | in decimal digits |
| float | 4 | 32 | +/- 3.4 \* 1038 | 6-7 |
| double | 8 | 64 | +/- 1.8 \* 10308 | 15 |

A floating-point literal can be denoted as a decimal point, a fraction part, an exponent (represented by E or e) and as an integer. We also add a suffix to the floating point literal as D, d, F or f.  The type of a floating-point literal defaults to double-precision floating-point.The following floating-point literals represent double-precision floating-point and floating-point values.

|  |  |
| --- | --- |
| 6.5E+32 (or 6.5E32) | Double-precision floating-point literal |
| 7D | Double-precision floating-point literal |
| .01f | Floating-point literal |

**String Literals**The string of characters is represented as String literals in Java. In Java a string is not a basic data type, rather it is an object. These strings are not stored in arrays as in C language. There are few methods provided in Java to combine strings, modify strings and to know whether to strings have the same value.  
We represent string literals as  
String myString = "How are you?";  
The above example shows how to represent a string. It consists of a series of characters inside double quotation marks.

Lets see some more examples of string literals:  
  
""    // the empty string  
"\""   // a string containing "  
"This is a string"   // a string containing 16 characters  
"This is a " +   // actually a string-valued constant expression,  
"two-line string"   // formed from two string literals

Strings can include the character escape codes as well, as shown here:  
String example = "Your Name, \"Sumit\"";  
System.out.println("Thankingyou,\nRichards\n");

**Null Literals**  
The final literal that we can use in Java programming is a Null literal. We specify the Null literal in the source code as 'null'. To reduce the number of references to an object, use null literal. The type of the null literal is always null. We typically assign null literals to object reference variables. For instance  
s = null;  
An this example an object is referenced by s. We reduce the number of references to an object by assigning null to s. Now, as in this example the object is no longer referenced so it will be available for the garbage collection i.e. the compiler will destroy it and the free memory will be allocated to the other object. Well, we will later learn about garbage collection.

**Data type:** The type of value that a variable can hold is called **data type**. When we declare a variable we need to specify the  type of value it will hold along with the name of the variable. This tells the compiler that the particular variable will hold certain amount of memory to store values. For example, in the lines of code above **num** is **int** type and takes two bytes to hold the integer value, **bol** is a **boolean** type and takes one bit to hold a boolean value .

Some common types of data types are used in the programming languages called as the primitive types like characters, integers,  floating point numbers etc. These primitive data types are given below where size represents the memory size it takes, For example, boolean takes a value **"true"/"false"** in 1 bit memory. It takes value **"false"** if not initialized (in the case of non-local variables)

**Java Primitive Data Types**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Type** | **Description** | **Size** | **Default Value** |
| boolean | true or false | 1-bit | false |
| char | Unicode Character | 16-bit | \u0000 |
| byte | Signed Integer | 8-bit | (byte) 0 |
| short | Signed Integer | 16-bit | (short) 0 |
| int | Signed Integer | 32-bit | 0 |
| long | Signed Integer | 64-bit | 0L |
| float | Real number | 32-bit | 0.0f |
| double | Real number | 64-bit | 0.0d |