# Unit 3

# Data types, Variables and Array

## *Java Is a Strongly Typed Language*

- It is important to state at the outset that Java is a strongly typed language.
- Indeed, part of Java's safety and robustness comes from this fact.
- First, every variable has a type, every expression has a type, and every type is strictly defined.
- Second, all assignments, whether explicit or via parameter passing in method calls, are checked for type compatibility.
- There are no automatic coercions or conversions of conflicting types as in some languages.
- The Java compiler checks all expressions and parameters to ensure that the types are compatible.
- Any type mismatches are errors that must be corrected before the compiler will finish compiling the class.

## *Java Basic Datatypes:*

Variables are nothing but reserve the memory location to store value. This means that when we create a variable to reserve some space in memory based. Based on data type of variable, the operating system allocate memory and decide what. Be told in reserve memory. Therefore, we can access different data types, variables you can store in teachers, decimals or characters in variables. There are two types of. Data type variables available in Java,

1. Primitive data types
2. Reference data types.
3.

## Primitive data type

There are eight primitive data types supported in Java. Primitive data types are predefined by language and name by keywords. Let us now look into details about the 8 primitive data type.

*Byte:*

- Byte data type is the 8 bit signed 2S complement integer.
- Minimum value is -128.
- Maximum value is 127.
- Default value is 0.
- Byte data type is used to save space in large array, mainly in place of integer.

*Short*

- Short data type is 16 bit sign 2S complement integer.
- The minimum value is -32,768.
- The maximum value is 32,767.
- It can also be used to save memory as a byte data type.
- A short is 2 times smaller than INT.
- Default value is 0.

Example:

```
Short S = 1000;

Short r = -200;
```

*Int*

- Int data type is 32 bit signed to a compliment integer.
- Minimum value is minus 2147483684.
- Maximum value is 2147483647.
- It is generally used as default data type for integer value.
- Default value is 0.

Example:

```
Int s =10000;

Int r = 8837;
```

*Long*

- Long data type is a 64 bit sign to S complement in teaser.
- Maximum value is -9223372036854775808.
- Maximum value is  9223372036854775808.
- It is used when a wide range than it is needed.
- Default value is 0.

Examples:

INT A = 10001.

INT B =2000001.

## *Float:*

- It is single precision 32 bit.
- It is mainly used to save memory in large array of floating point numbers.
- Minimum value is 35.1 F.
- Maximum value is 35.154.
- Default Value is 0.0 F.

Example,

Float f = 345.5F

## *Double*

- Double data type is a double precision 64 bit this.
- The type is generally used as default data type for decimal value.
- It should never be used for precise value, such as currency default.
- Default value is 0.0 D.

Example;

double d1 = 121.25;

## *Boolean*

- It represents one bit of information there.
- There are only possible values, true and false.
- This is used for true/false conditions.
- Default value is false.

Example:

Boolean one = true;

## *Char*

- It is a single 16 bit unique code character.
- Minimum value is 0.
- Maximum value is 65535.
- It is used to store any character.

Example,

Char letter = ‘A’;

## *Literals in Java*

Literals have a constant value and a fixed data type and are often known as constants in Java. Literals are assigned to a variable to provide a value to the variable.

An example of assigning a literal to a variable is:

int age = 21;

In the above code snippet, int is the datatype, age is the variable name and 21 is the literal. So, we can say that the literal is providing the variable with a value of 21.

## Types of Literals in Java

Generally, there are 5 types of literals that can be further expanded into various other literals. We will take a look at them vividly in this section.

The five main Literal types are:

1. Integer Literal
2. Float/Double Literal
3. Character Literal
4. Boolean Literal
5. String Literal

### *Integer Literal:*

Integer literals are basically a number sequence that doesn’t contain any decimal point in between them.

There are four types of Integer Literals:

*1. Decimal Integer:*

They are integers having a base value of 10; i.e; containing values between 0 to 9. It can be a positive value(+) or a negative value(-) but it cannot contain any point in between them. Example: 1000, 1234, +78, -82, etc.

```
int decimal_int=1234;
```

*2. Octal Integer:*

They are integers having a base value of 8; i.e; containing values between 0 to 7. All octal numbers must start with a 0. Example: 012, 077,075, etc.

```
int octal_int=077;
```

*3. Hexadecimal Integer:*

They are integers having a base value of 16; i.e; containing values between 0 to 15. It is a special type of integer that contains both digits as well as characters. The digits range from 0 to 9 and the numbers 10 to 15 are replaced by characters a to f. Any integer starting with 0x or 0X is considered to be a hexadecimal integer. Example: 0xff, 0x2a, 0xf1f2, etc.

```
int hexadec_int=0x1ff2;
```

*4. Binary Integer:*

They are integers with a base value of 2; i.e; contains only two digits 0 and 1. Binary integers start with a 0b indicating that it is a binary digit. Example: 0b100101, 0b1010101, etc.

```
int binary_int=0b1010101;
```

Program Showing all the integer literals:

```
package com.DataFlair.Literals;
public class Literals_Integers
{
   public void main()
   {
      int decimal_int=1234;
      int octal_int=077;
      int hexadec_int=0x1ff2;
      int binary_int=0b1010101;
      System.out.println("This is a Decimal Literal: "+decimal_int);
      System.out.println("This is an Octal Literal: "+octal_int);
      System.out.println("This is a Hexa Decimal Literal: "+hexadec_int);
      System.out.println("This is a Binary Literal: "+binary_int);
   }
}
```

The output of the above code:

```
This is a Decimal Literal: 1234
This is an Octal Literal: 63
This is a Hexa Decimal Literal: 8178
This is a Binary Literal: 85
```

## *Floating Point Literal:*

Floating-point literals are values that contain a decimal point in between them. Floating-point literals are generally double data type by default. We can assign them to float data types by adding an f at the end of the value.

Example of declaring a float literal:

```
float val_float=1.7732f;
```

Example of declaring a double literal:

```
float val_double=1.7732; //By default the compiler assigns double datatype.
float val_double=1.7732d;
double val_double=1.7732;
```

Program Showing all the floating literals:

```
package com.DataFlair.Literals;
public class Literals_Float
{
   public void main()
   {
      float val_float=1.7732f;
      double val_double=1.7732d;
      float val_exponent=123E4f;
      System.out.println("This is a Floating Point Literal"+val_float);
      System.out.println("This is a Decimal Literal"+val_double);
      System.out.println("This is an Exponential Literal"+val_exponent);
   }
}
```

The output of the above code:

This is a Floating Point Literal1.7732

This is a Decimal Literal1.7732

This is an Exponential Literal1230000.0

## *Boolean Literal:*

A boolean literal is a literal that contains only two values true and false. It is declared using the keyword boolean. It is a very useful literal to declare flag variables in different programs to terminate a looping sequence.

Program showing the boolean Literals:

```
package com.DataFlair.Literals;

public class Literals_boolean

{

   public void main()

   {

      boolean flag1=true;

      boolean flag2=false;

      System.out.println("This is a boolean true flag variable"+flag1);

      System.out.println("This is a boolean false flag variable"+flag2);

   }

}
```

The output of the above code:

This is a boolean true flag variabletrue

This is a boolean false flag variablefalse

## *String Literal:*

A string is basically an array of characters. In java, we have a special class for strings that allows users to implement strings to a program very easily. Anything written inside a double quote is a string “”.

Example of String Literal:

String Company = “DataFlair”;//Valid String Literal

String Company = DataFlair;//Invalid as there is no double quote.

## *Character Literal:*

Character Literals in java are represented with a single quote. The general difference between string literal and character literal is that character literal contains only one character whereas string literal contains a set of characters.

A character literal can be represented in four ways:

*1.Single quote character:*

Example:

char ch = ‘A’;

*2. Character Literal as an Integer Literal:*

Example:

char number = 0065;

*3. Unicode representation of character Literal:*

Example:

char uni = ‘\u0065’

## *Variable*

A variable is identifier used to store single data item. This data could be a numerical quantity of characters, constants during execution of program data must be assigned to a variable. Therefore, a variable can take different values at a different time during the execution of program data. Stored in variables can be accessed at the time by referring to the variable name.

## *Variable Declaration*

There are two ways to declare a variable in Java. The first method is to assign the initial value to the variable. The second method declares variable without initial value.

Declare a Variable with Initial Value

Data_type variable_name = value;

Data_type variable_name;

Example;

```
public class CreateVariable {
   public static void main(String[] args)
         {
              //variable declaration
               int student_id = 10;
               String student_name = "Java coder";
               double numbers = 3.21;
               Boolean shows = true;
               System.out.println("Name:" +student_name+ "\nAge:" +student_id);
               System.out.println("Number:" +numbers+ "\nBoolean:" +shows);
        }
 }
```

## *Java Variables - Dynamic Initialization*

Initialization is the process of providing value to a variable at declaration time. A variable is initialized once in its life time. Any attempt of setting a variable's value after its declaration is called assignment. To use a local variable you have to either initialize or assign it before the variable is first used. But for class members, the compulsion is not so strict. If you don't initialize them then compiler takes care of the initialization process and set class members to default values.

Java allows its programmers to initialize a variable at run time also. Initializing a variable at run time is called dynamic initialization. The following piece of code (DynamicInitializationDemo.java) demonstrates it.

```
/* DynamicInitializationDemo.java */
public class DynamicInitializationDemo
{
   public static void main(String[] args)
   {
      //dynSqrt will be initialized when Math.sqrt
      //will be executed at run time
      double dynSqrt = Math.sqrt (16);
      System.out.println("sqrt of 16 is : " + dynSqrt);
   }
}
```

OUTPUT

```
======
sqrt of 16 is : 4.0
```

## *Scope of a Variable*

In programming, a variable can be declared and defined inside a class, method, or block. It defines the scope of the variable i.e. the visibility or accessibility of a variable. Variable declared inside a block or method are not visible to outside. If we try to do so, we will get a compilation error. Note that the scope of a variable can be nested.

- We can declare variables anywhere in the program but it has limited scope.
- A variable can be a parameter of a method or constructor.
- A variable can be defined and declared inside the body of a method and constructor.
- It can also be defined inside blocks and loops.
- Variable declared inside main() function cannot be accessed outside the main() function

Types of Variables and its Scope

There are three types of variables.

1. Instance Variables
2. Class Variables
3. Local Variables

*Instance Variables*

A variable which is declared inside a class and outside all the methods and blocks is an instance variable. The general scope of an instance variable is throughout the class except in static methods. The lifetime of an instance variable is until the object stays in memory.

*Class Variables*

A variable which is declared inside a class, outside all the blocks and is marked static is known as a class variable. The general scope of a class variable is throughout the class and the lifetime of a class variable is until the end of the program or as long as the class is loaded in memory.

*Local Variables*

All other variables which are not instance and class variables are treated as local variables including the parameters in a method. Scope of a local variable is within the block in which it is declared and the lifetime of a local variable is until the control leaves the block in which it is declared.

Example

```
public class scope_and_lifetime {
   int num1, num2;   //Instance Variables
   static int result;  //Class Variable
   int add(int a, int b){  //Local Variables
      num1 = a;
      num2 = b;
      return a+b;
   }
   public static void main(String args[]){
      scope_and_lifetime ob = new scope_and_lifetime();
      result = ob.add(10, 20);
      System.out.println("Sum = " + result);
   }
}
```

## *Type conversions Java*

Java provides various datatypes to store various data values. It provides 7 primitive datatypes (stores single values) as listed below −

boolean − Stores 1-bit value representing true or, false.

byte − Stores twos compliment integer up to 8 bits.

char − Stores a Unicode character value up to 16 bits.

short − Stores an integer value upto 16 bits.

int − Stores an integer value upto 32 bits.

long − Stores an integer value upto 64 bits.

float − Stores a floating point value upto 32bits.

double − Stores a floating point value up to 64 bits.

Type Casting/type conversion

Converting one primitive datatype into another is known as type casting (type conversion) in Java. You can cast the primitive datatypes in two ways namely, Widening and, Narrowing.

Widening − Converting a lower datatype to a higher datatype is known as widening. In this case the casting/conversion is done automatically therefore, it is known as implicit type casting. In this case both datatypes should be compatible with each other.

Example

```
public class WideningExample {
   public static void main(String args[]){
      char ch = 'C';
      int i = ch;
      System.out.println(i);
   }
}
```

Output

```
Integer value of the given character: 67
```

Narrowing − Converting a higher datatype to a lower datatype is known as narrowing. In this case the casting/conversion is not done automatically, you need to convert explicitly using the cast operator “( )” explicitly. Therefore, it is known as explicit type casting. In this case both datatypes need not be compatible with each other.

Example

```
import java.util.Scanner;
public class NarrowingExample {
   public static void main(String args[]){
      Scanner sc = new Scanner(System.in);
```

```
    System.out.println("Enter an integer value: ");
    int i = sc.nextInt();
    char ch = (char) i;
    System.out.println("Character value of the given integer: "+ch);
  }
}
```

Output

```
Enter an integer value:
67
Character value of the given integer: C
```

## *Array in Java*

An array is an structured data type consisting of group of element of the same type in other words an type of variable multiple values. Array is special can store multiple values.

Java air is a object which contains elements of similar data types. The element of arrays are stored in location. It is data structure which store the similar elements. We can only store fix set of elements in Java elements.

→ Type of Array in Java :

i) One dimensional Array
ii) Multi dimensional Array

## One dimensional array

Single dimensional array use single index to store elements. You can get all the elements of array by just increment its index by one.

Syntax to Declare One dimensional Array in Java

```
dataType[] arr;

dataType arr[];
```

Example;

```
Int[] arr;

Float[] new_array;
```

## Multi-Dimensional Array

A multi-dimensional array is very much similar to a single dimensional array. It can have multiple rows and multiple columns unlike single dimensional array, which can have only one row index.

It represent data into tabular form in which data is stored into row and columns.

Syntax to Declare multidimensional Array in Java

```
datatype[row][column ] arrayName;
```

Example;

int[5][5] new_array;

char[2][2] input;

## Alternative Array Declaration Syntax

There is a second form that can be used to declare an array:

type[ ] var-name;

Here, the square brackets follow the type specifier, not the name of the array variable. For example, the following two declarations are equivalent:

int counter[] = new int [3];

int[] counter = new int [3];

The following declarations are also equivalent:

char table [] [] = new char [3] [4];

char [] [] table = new char [3] [4];

## *Scanner Object*

Scanner class in Java is found in the java.util package. Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.

The Java Scanner class breaks the input into tokens using a delimiter which is whitespace by default. It provides many methods to read and parse various primitive values.

The Java Scanner class is widely used to parse text for strings and primitive types using a regular expression. It is the simplest way to get input in Java. By the help of Scanner in Java, we can get input from the user in primitive types such as int, long, double, byte, float, short, etc.

To get the instance of Java Scanner which reads input from the user, we need to pass the input stream (System.in) in the constructor of Scanner class. For Example:

```
Scanner in = new Scanner(System.in);
```

Example:

```
import java.util.*;

public class ScannerExample {

public static void main(String args[]){

        Scanner in = new Scanner(System.in);

        System.out.print("Enter your name: ");

        String name = in.nextLine();

        System.out.println("Name is: " + name);

        in.close();

        }

}
```

Scanner Input method for different datatypes:

| | | |
|---|---|---|
| Boolean | nextBoolean() | It scans the boolean value and returns that value. |
| byte | nextByte() | It scans the next token of the input as a byte. |
| double | nextDouble() | It scans the next token of the input as a double. |
| float | nextFloat() | It scans the next token of the input as a float. |
| int | nextInt() | It scans the next token of the input as an Int. |

| | | |
|---|---|---|
| String | nextLine() | It is used to get the input string that was skipped of the Scanner object. |
| long | nextLong() | It scans the next token of the input as a long. |
| short | nextShort() | It scans the next token of the input as a short. |