1.4.7. Data Types in Java

Data type is a special keyword used to assignenough memory space for the data, in other words Data type is used for on behalf of the data in main memory (RAM) of the computer.

In java, there are two types of data types

- Primitive data types
- Non-primitive data types

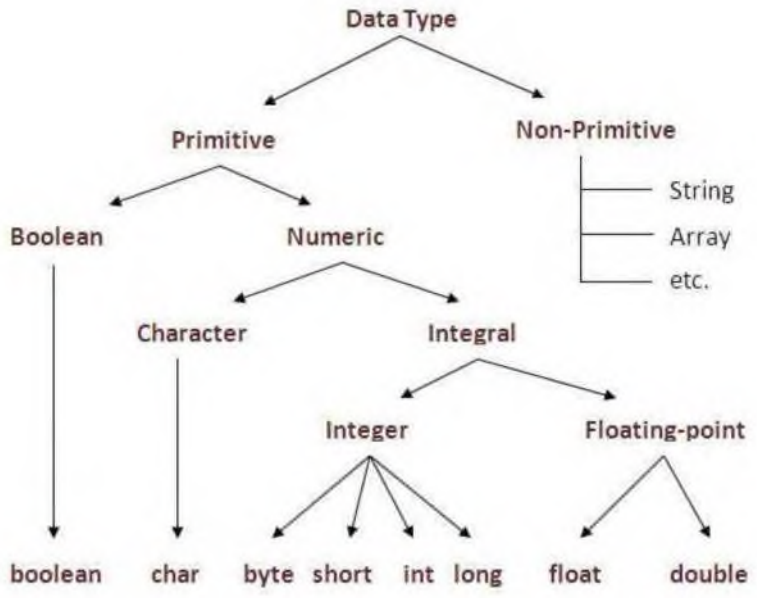

Figure 1: Classification of Data type

1.Primitive data types

Primitive data types are variables permits us to **store only one value** but they not at all allows us to store multiple values of similar type. This is a data type whose variable can contain maximum one value at a time.

Example:

```
int a; // valid

a=10; // valid
a=10, 20, 30; // invalid
```

There are eight primitive data types given by Java. Primitive data types are predefined by the language and named through a keyword.

- **Numeric primitives:**short, int, long, float and double. These primitive data types can

contain only numeric data. Operations related with such data types are those of easy arithmetic (addition, subtraction, etc.) or of comparisons (is greater than, is equal to, etc.)
Example: double a=120.20;

double b=50.20;
double c= a+b;

- **Textual primitives:**byte and char. These primitive data types contain characters (that can be Unicode alphabets or even numbers). Operations related with such types are those of textual operation (comparing two words, joining characters to make words, etc.). though, byte and char can also support arithmetic operations. Example: char a='A';
  char b='B';
- **Boolean and null primitives:** boolean and null.
  Example: boolean flag=TRUE ;

Data Type	Default Value	Default size	Range
boolean	false	1 bit	True or False only
char	'\u0000'	2 byte	0 to 65,535
byte	0	1 byte	-128 to 127
short	0	2 byte	-32,768 to 32,767
int	0	4 byte	-2,147,483,648 to 2,147,483,647
long	0L	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	0.0f	4 byte	1.40129846432481707e-45 to 3.40282346638528860e+38
double	0.0d	8 byte	4.94065645841246544e-324d to 1.79769313486231570e+308d

Table : Primitive Data type with Memory Size

2. Non-Primitive Data Types

It is Used to **store multiple values**.Reference variables are produced using defined constructors of the **classes**. They are used to contact objects. These variables are declared to be of a particular type that cannot be changed. Class objects and various types of array variables come below reference data type. Default value of any reference variable is null. A reference variable can be used to refer any other object of the declared type or any compatible type.

Objects and **Arrays** are the reference or non-primitive data types in Java. They are so called since they are handled "by reference" i.e. variables of their kind store the address of the object or array is stored in a variable. They are passed by reference. For Ex:

char [] arr = { 'a', 'b', 'c', 'd' }; //'arr' stores the references for the 4 values

Simple Example for Data type:

```
public class PrimitiveDemo
{
    public static void main(String[] args)
    {
```

```
        byte b =100;
        short s =123;
        int v = 123543;
        int calc = -9876345;
        long amountVal = 1234567891;
        float intrestRate = 12.25f;
        double sineVal = 12345.234d;
        boolean flag = true;
        boolean val = false;
        char ch1 = 88; // code for X
        char ch2 = 'Y';
        System.out.println("byte Value = "+ b);
        System.out.println("short Value = "+ s);
        System.out.println("int Value = "+ v);
        System.out.println("int second Value = "+ calc);
        System.out.println("long Value = "+ amountVal);
        System.out.println("float Value = "+ intrestRate);
        System.out.println("double Value = "+ sineVal);
        System.out.println("boolean Value = "+ flag);
        System.out.println("boolean Value = "+ val);
        System.out.println("char Value = "+ ch1);
        System.out.println("char Value = "+ ch2);
    }

}
```

Output:

```
byte Value = 100 short Value = 123 int Value = 123543 int second Value = -
9876345
long Value = 1234567891
float Value = 12.25
double Value = 12345.234
boolean Value = true
boolean Value = false char Value = X char Value = Y
```

The Primitive Types

The primitive types are also commonly referred to as simple types.Java defines eight primitive types of data: byte, short, int, long, char, float, double, and boolean.

These can be put in four groups:

Integers This group includes byte, short, int, and long, which are for whole-valued signed numbers.

Floating-point numbers This group includes float and double, which represent numbers with fractional precision.

Characters This group includes char, which represents symbols in a character set, like letters and numbers.
Boolean This group includes boolean, which is a special type for representing true/false values.

Types of Literals:
a. Integer Literals
b. Floating-Point Literals
c. Boolean Literals
d. Character Literals
e. String Literals

The Java Keywords:
There are 50 keywords currently defined in the Java language.
These keywords, combined with the syntax of the operators and separators, form the foundation of the Java language.
These keywords cannot be used as names for a variable, class, or method.

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Table : Java Keywords

1.4.8. VARIABLES

The variable is the fundamental unit of storage in a Java program. A variable is defined by the mixture of an identifier, a type, and an optional initializer. All variables have a range, which defines their visibility, and a lifetime. In Java, all variables should be declared before they can be used. The fundamental form of a variable declaration is shown here:

type identifier [= value][, identifier [= value] ...] ;

- type can be any one java data type, or the name of a class or interface.
- The identifier is the name of the variable.
- You can initialize the variable by mentioning an equal sign and a value.
- The initialization expression must answer in a value of the same type as that mentioned for the variable.

If we want to declare more than one variable of the specified type, we can use a comma separated list. Here are several examples of variable declarations of various types.

int a, b, c; **// declares three ints, a, b, and c.** 6

```
int d = 3, e, f = 5; // declares three more ints, initializing d and f.
byte z = 22; // initializes z.
```

Dynamic Initialization

Java permits variables to be initialized dynamically, any expression valid at the time the variable is declared.

Example Program: Demonstrate dynamic initialization.

```
//Demonstrate dynamic initialization.
class DynInit
{
public static void main(String args[])
{
double a = 3.0, b = 4.0;
// c is dynamically initialized

double c = Math.sqrt(a * a + b * b);
System.out.println("Hypotenuse is " + c);
}
}
```

The Scope and Lifetime of Variables

Scope:

- Each variables used have been declared at the beginning of the main( ) method. Java allows variables to be declared inside any block. A block is started with an opening curly brace and closed by a closing curly brace.
- A block defines a scope. Thus, every time you begin a new block, you are creating a new scope. A scope decides what objects are visible to other places of your program. It also decides the lifetime of those objects.
- Many other programming languages define two general types of scopes: global and local. In Java, there are two major scopes are defined in the class and those defined by a function. Variables declared within a scope are not visible (that is, accessible) to program that is given outside the scope.
- Nesting of scopes is possible. For example, Every time you create a block of code, you can create a new, nested scope. Whenever it happens the outer scope encloses the internal scope.

This means that objects declared on the outside scope will be visible to code within the internal scope.

Example Program: Scope of Variables

```
// Demonstrate block scope.
class Scope
{
public static void main(String args[])
{
int x; // known to all code within main
x = 10;
if(x == 10)
```

```
{
// start new scope
int y = 20;
// known only to this block 7
// x and y both known here.
System.out.println("x and y: " + x + " " + y);
x = y * 2;
}
// y = 100; // Error! y not known here
// x is still known here.
System.out.println("x is " + x);
}
}
```

Lifetime:

A variable declared inside a block will drop its value when the block is left. Thus, the life span of a variable is limited to its scope. If a variable declaration have aninitialize, then that variable can be reinitializedevery time and the block in which it is stated.

Example Program : Lifetime of a variable

```
// Demonstrate lifetime of a variable.
class LifeTime
{
public static void main(String args[])
{
int x;
for(x = 0; x < 3; x++)
{
int y = -1; // y is initialized each time block is entered
System.out.println("y is: " + y); // this always prints -1
y = 100;
System.out.println("y is now: " + y);
}
}
}
```

Output:

```
y is: -1
y is now: 100
y is: -1
y is now: 100
y is: -1
y is now: 100
```