**Language Basics**

### Data Types in Java

Java language has a rich implementation of data types. Data types specify size and the type of values that can be stored in an identifier.

In java, data types are classified into two catagories :

1. Primitive Data type
2. Non-Primitive Data type

#### 1) Primitive Data type

A primitive data type can be of eight types :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Primitive Data types | | | | | | | |
| char | boolean | byte | short | int | long | float | double |

Once a primitive data type has been declared its type can never change, although in most cases its value can change. These eight primitive type can be put into four groups

#### Integer

This group includes byte, short, int, long

**byte :** It is 1 byte(8-bits) integer data type. Value range from -128 to 127. Default value zero. example: byte b=10;

**short :** It is 2 bytes(16-bits) integer data type. Value range from -32768 to 32767. Default value zero. example: short s=11;

**int :** It is 4 bytes(32-bits) integer data type. Value range from -2147483648 to 2147483647. Default value zero. example: int i=10;

**long :** It is 8 bytes(64-bits) integer data type. Value range from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Default value zero. example: long l=100012;

#### Floating-Point Number

This group includes float, double

**float :** It is 4 bytes(32-bits) float data type. Default value 0.0f. example: float ff=10.3f;

**double :** It is 8 bytes(64-bits) float data type. Default value 0.0d. example: double db=11.123;

#### Characters

This group represent char, which represent symbols in a character set, like letters and numbers.

**char :** It is 2 bytes(16-bits) unsigned unicode character. Range 0 to 65,535. example: char c='a';

#### Boolean

This group represent boolean, which is a special type for representing true/false values. They are defined constant of the language. example: boolean b=true;

#### 2) Non-Primitive(Reference) Data type

A reference data type is used to refer to an object. A reference variable is declare to be of specific and that type can never be change. We will talk a lot more about reference data type later in Classes and Object lesson.

### Identifiers in Java

All Java components require names. Name used for classes, methods, interfaces and variables are called **Identifier**. Identifier must follow some rules. Here are the rules:

* All identifiers must start with either a letter( a to z or A to Z ) or currency character($) or an underscore.
* After the first character, an identifier can have any combination of characters.
* A Java **keyword** cannot be used as an identifier.
* Identifiers in Java are case sensitive, foo and Foo are two different identifiers.

### Type Casting

Assigning a value of one type to a variable of another type is known as **Type Casting**.

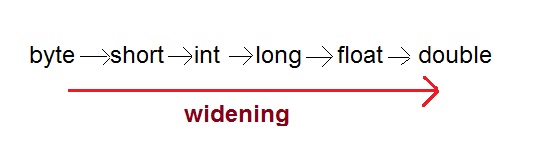
**Example :**

int x = 10;

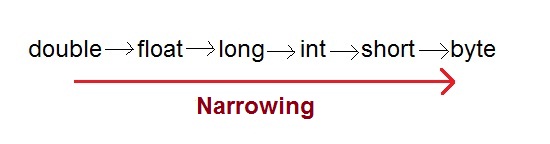
byte y = (byte)x;

In Java, type casting is classified into two types,

* Widening Casting(Implicit)



* Narrowing Casting(Explicitly done)



#### Widening or Automatic type converion

Automatic Type casting take place when,

* the two types are compatible
* the target type is larger than the source type

**Example :**

public class Test

{

public static void main(String[] args)

{

int i = 100;

long l = i; //no explicit type casting required

float f = l; //no explicit type casting required

System.out.println("Int value "+i);

System.out.println("Long value "+l);

System.out.println("Float value "+f);

}

}

Int value 100 Long value 100 Float value 100.0

#### Narrowing or Explicit type conversion

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

**Example :**

public class Test

{

public static void main(String[] args)

{

double d = 100.04;

long l = (long)d; //explicit type casting required

int i = (int)l; //explicit type casting required

System.out.println("Double value "+d);

System.out.println("Long value "+l);

System.out.println("Int value "+i);

}

}

Double value 100.04 Long value 100 Int value 100

## Variable

### What is a variable?

When we want to store any information, we store it in an address of the computer. Instead of remembering the complex address where we have stored our information, we name that address.The naming of an address is known as variable. Variable is the name of memory location.

Java Programming language defines mainly three kind of variables.

1. Instance variables
2. Static Variables
3. Local Variables

#### 1) Instance variables

Instance variables are variables that are declare inside a class but outside any method,constructor or block. Instance variable are also variable of object commonly known as field or property. They are referred as object variable. Each object has its own copy of each variable and thus, it doesn't effect the instance variable if one object changes the value of the variable.

class Student

{

String name;

int age;

}

Here **name** and **age** are instance variable of Student class.

#### 2) Static variables

Static are class variables declared with static keyword. Static variables are initialized only once. Static variables are also used in declaring constant along with final keyword.

class Student

{

String name;

int age;

static int instituteCode=1101;

}

Here **instituteCode** is a static variable. Each object of Student class will share instituteCode property.

**Additional points on static variable:**

* static variable are also known as class variable.
* static means to remain constant.
* In Java, it means that it will be constant for all the instances created for that class.
* static variable need not be called from object.
* It is called by *classname.static variable name*

**Note:** A static variable can never be defined inside a method i.e it can never be a local variable.

*Example:*

Suppose you make 2 objects of class Student and you change the value of static variable from one object. Now when you print it from other object, it will display the changed value. This is because it was declared static i.e it is constant for every object created.

package studytonight;

class Student{

int a;

static int id = 35;

void change(){

System.out.println(id);

}

}

public class StudyTonight {

public static void main(String[] args) {

Student o1 = new Student();

Student o2 = new Student();

o1.change();

Student.id = 1;

o2.change();

}

}

OutPut:-

35

1

#### 3) Local variables

Local variables are declared in method, constructor or block. Local variables are initialized when method, constructor or block start and will be destroyed once its end. Local variable reside in stack. Access modifiers are not used for local variable.

float getDiscount(int price)

{

float discount;

discount=price\*(20/100);

return discount;

}

Here **discount** is a local variable.

### Concept of Array in Java

An array is a collection of similar data types. Array is a container object that hold values of homogenous type. It is also known as static data structure because size of an array must be specified at the time of its declaration.

An array can be either primitive or reference type. It gets memory in heap area. Index of array starts from zero to size-1.

#### Features of Array

* It is always indexed. Index begins from 0.
* It is a collection of similar data types.
* It occupies a contiguous memory location.

#### Array Declaration

**Syntax :**

datatype[] identifier;

or

datatype identifier[];

Both are valid syntax for array declaration. But the former is more readable.

**Example :**

int[ ] arr;

char[ ] arr;

short[ ] arr;

long[ ] arr;

int[ ][ ] arr; // two dimensional array.

#### Initialization of Array

new operator is used to initialize an array.

**Example :**

int[] arr = new int[10]; //this creates an empty array named arr of integer type whose size is 10.

or

int[] arr = {10,20,30,40,50}; //this creates an array named arr whose elements are given.

#### Accessing array element

As mention ealier array index starts from 0. To access nth element of an array. Syntax

arrayname[n-1];

*Example :* To access 4th element of a given array

int[ ] arr = {10,20,30,40};

System.out.println("Element at 4th place" + arr[3]);

The above code will print the 4th element of array arr on console.

**Note:** To find the length of an array, we can use the following syntax: array\_name.length. There are no braces infront of length. Its not length().

#### foreach or enhanced for loop

J2SE 5 introduces special type of for loop called foreach loop to access elements of array. Using foreach loop you can access complete array sequentially without using index of array. Let us see an example of foreach loop.

class Test

{

public static void main(String[] args)

{

int[] arr = {10, 20, 30, 40};

for(int x : arr)

{

System.out.println(x);

}

}

}

OutPut:--

10 20 30 40

### Java Operators

Java provides a rich set of operators environment. Java operators can be divided into following categories:

* Arithmetic operators
* Relation operators
* Logical operators
* Bitwise operators
* Assignment operators
* Conditional operators
* Misc operators

#### Arithmetic operators

Arithmetic operators are used in mathematical expression in the same way that are used in algebra.

|  |  |
| --- | --- |
| Operator | Description |
| + | adds two operands |
| - | subtract second operands from first |
| \* | multiply two operand |
| / | divide numerator by denumerator |
| % | remainder of division |
| ++ | Increment operator increases integer value by one |
| -- | Decrement operator decreases integer value by one |

#### Relation operators

The following table shows all relation operators supported by Java.

|  |  |
| --- | --- |
| Operator | Description |
| == | Check if two operand are equal |
| != | Check if two operand are not equal. |
| > | Check if operand on the left is greater than operand on the right |
| < | Check operand on the left is smaller than right operand |
| >= | check left operand is greater than or equal to right operand |
| <= | Check if operand on left is smaller than or equal to right operand |

#### Logical operators

Java supports following 3 logical operator. Suppose a=1 and b=0;

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| && | Logical AND | (a && b) is false |
| || | Logical OR | (a || b) is true |
| ! | Logical NOT | (!a) is false |

#### Bitwise operators

Java defines several bitwise operators that can be applied to the integer types long, int, short, char and byte

|  |  |
| --- | --- |
| Operator | Description |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise exclusive OR |
| << | left shift |
| >> | right shift |

Now lets see truth table for bitwise &, | and ^

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | b | a & b | a | b | a ^ b |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

The bitwise shift operators shifts the bit value. The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value are to be shifted. Both operands have the same precedence.

**Example:**

a = 0001000

b = 2

a << b = 0100000

a >> b = 0000010

#### Assignment Operators

Assignment operator supported by Java are as follows:

|  |  |  |
| --- | --- | --- |
| Operator | Description | Example |
| = | assigns values from right side operands to left side operand | a = b |
| += | adds right operand to the left operand and assign the result to left | a+=b is same as a=a+b |
| -= | subtracts right operand from the left operand and assign the result to left operand | a-=b is same as a=a-b |
| \*= | mutiply left operand with the right operand and assign the result to left operand | a\*=b is same as a=a\*b |
| /= | divides left operand with the right operand and assign the result to left operand | a/=b is same as a=a/b |
| %= | calculate modulus using two operands and assign the result to left operand | a%=b is same as a=a%b |