**Operator Precedence in Java:**

* Java has well-defined rules for specifying the order in which the operators in an

expression are evaluated when the expression has several operators.

* For example, multiplication and division have a higher precedence than addition and subtraction. Precedence rules can be overridden by explicit parentheses.

**Precedence Order:**

* When two operators share an operand the operator with the higher precedence goes first.
* For example, 1+2\*3 is treated as 1+ (2\*3), whereas 1 \* 2 + 3 is treated as (1 \* 2) + 3 since multiplication has a higher precedence than addition.

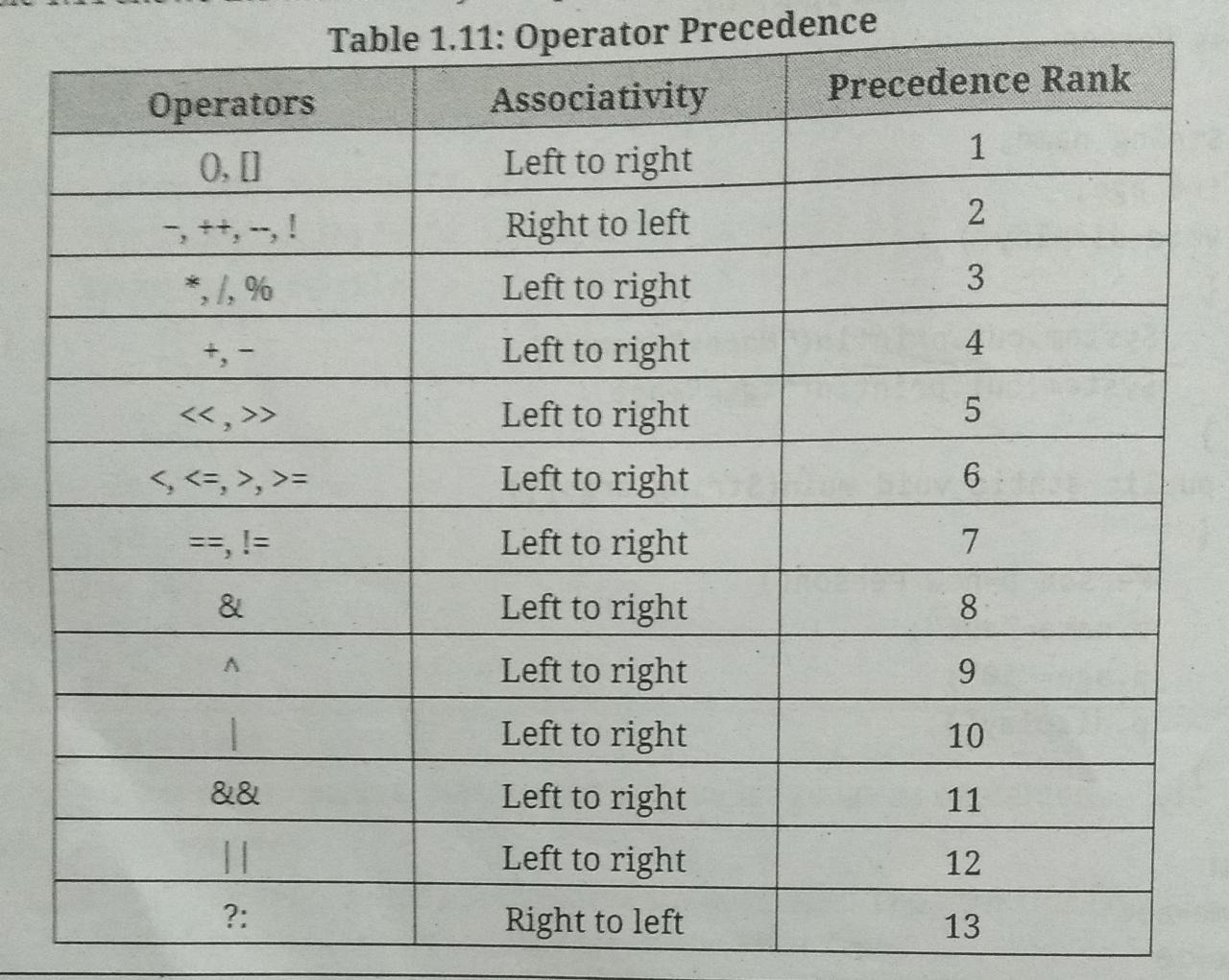
**Operator Associativity in Java:**

• When two operators with the same precedence the expression is evaluated according to its associativity.

•For example, x = y = z = 17 is treated as x = (y = (z = 17)), leaving all three variables with the value 17, since the operator has right-to-left associativity (and an assignment statement evaluates to the value on the right hand side).

•On the other hand, 72/2/3 is treated as (72/2)/3 since the / operator has left-to-right associativity.

Table shows the associativity and precedence order of java operators.



**Variables**

• Variables are containers for storing data values. Variables are the basic units used to store any information in Java.

• A variable is an identifier that denotes a storage location used to store a data value.

The value in the variable can be changed during the program's execution.

• Variables are nothing but reserved memory locations to store values. This means that when we create a variable we reserve some space in memory.

• Following are the some rules used for variable naming:

1. Variable name must begin with either a letter or the dollar sign "$" or the underscore character "\_".

2. Variable names are case-sensitive. Upper case and lower case must be distinct this means that the variable Abc is not the same as abc or ABC.

3. Variable must not begin with a digit.

4. White space is not allowed in variable naming.

5. Variable name should not be a keyword.

6. Variable names can be of any length.

**Variable Declaration:**

• In Java, all the variables must be declared before use. A variable declaration, in its simplest form, includes the name and the data type of variables.

• Declaration does following three things:

1. It tells variable name to the compiler.

2. It specifies the data type of the data hold by variable.

3. The place of declaration in the program decided the scope of the variable.

• Following is the basic **form/syntax** of a variable declaration:

data type variable\_Name;

Here, data type is one of Java's data types and variable Name is the name of the

variable.

• To declare more than one variable of the specified type, you can use a comma-separated list as given below:

data\_type variable\_Name1, variable\_Name2, ….variable\_NameN;

Example:

int age; // Declares age

int a, b, c; // Declares three ints, a, b, and c.

**Variable Initialization:**

• Variable initialization means assigning a value to variables. Initializing a variable means specifying an initial value to assign to it (i.e., before it is used at all).

• In Java, we can assign a value to variables in two ways:

**1. Static:** This means that the memory is determined for variables when the program starts. Variables can be assigned initial values at the time of declaration.

Assignment operator (=) assigns the value of an expression to a variable.

Syntax: variable\_name-value; OR data\_type variable\_name-value;

Examples:

IntitalValue = 0;

a=b=c=0;

int a=10, b= 10; // Variable initialization

byte B =22; // initializes a byte type variable 8.

double pi 3.14159; // declares and assigns a value of PI.

.

**2. Dynamic:** Dynamic means that in Java, we can declare variables anywhere in theprogram, because when the statement is executed the memory is assigned to them.

Java allows its programmers to initialize a variable at run time also. Initializing avariable at run time is called dynamic initialization.

Program for dynamic initialization of variables.

class DynamicInit{

public static void main(String args[]){

double a = 3.0, b=4.0;

//c is dynamically initialised

double c = Math.sqrt(a \* a + b \* b);

System.out.println(“C = ”+c);

}

Output:

C=5.0

**Scope of Variables:**

• The area of the program where the variable is accessible is called its scope. Scope refers to the lifetime and accessibility of a variable.

• The scope of a variable defines the section of the code in which the variable is accessible or visible.

• Scope is the lifetime of a variable and refers to how long the variable exists before it is destroyed.

Scope itself specifies that the system allocates and de-allocates the memory for the variable.

• Java variables are actually classified into three categories and their scope is defined according their category as given below:

**1. Instance Variables:** Instances are declared inside the class. Instance variables are created when the objects are instantiated (created) and those are associated with the objects. They take different values for each object. Instance variables are declared in a class, but outside a method, constructor or any block. The instance variables are visible for all methods, constructors and block in the class.

**2. Class/Static Variables:** Class variables are declared inside the class and global to the class and belong to the entire set of object and that class creates. Only one memory location is created for each class variable. Visibility is similar to instance variables.

**3. Local variables:** Local variables declared and used inside the methods. They are not available outside of the method definition. Local variables can be declared inside the program blocks that are defined between the opening { and closing braces}.These variables are visible to

the program control leaves block, all the variables in the block will cease to exit only. When the program control leaves block ,all the variables in the block will cease to exit

Program for scope of variables.

public class Sample

{

string name; //instance variable

int age;

static int count; //class variable

void accept(string n, int a) // n, a are local variables

{

name=n;

age=a;

}

void show()

{

count++;

System.out.println("count="+count);

void display()

{

System.out.println("Name="+name);

System.out.println("Age="+age);

}

public static void main(String args[])

{

Sample s1=new Sample();

Sample s2=new Sample();

s1.accept("abc", 10);

s1.display();

s1.show();

s2.accept("xyz", 20);

s2.display();

s2.show();

Output :

Name=abc

Age-10

count=1

Name=xyz

Age=20

count=2

**Final Variable**

• Variables are useful when we need to store information that can be changed as program runs.

• However, there may be certain situations in the program in which the value of variable should not be allowed to modify or change. This is accomplished using a special type of variable known as final variable.

• The final variable also called constant variable. Final variable is a variable with a value that cannot be modified during the execution of the program.

• To declare a final variable, use the final keyword before the variable declaration and initialize it with a value. Its syntax is,

final data\_type varName = value;

For example:

final double PI 3.141592;

• Above statement declares a variable PI of type double which is initialized to 3.141592, that cannot be modified. If an attempt is made to modify a final variable after it is initialized, the compiler issues an error message "cannot assign a value to final variable PI

Program to find area of circle using final variables.

public class area{

public static void main(String args[])

{

int r = 10;

final double PI = 3.141592;

double area;

area = PI \* r \* r;

System.out.println(“Area of Circle is : ”+area);

}

}

Output : 314.1592