Core Java – Language Basic

Fresher Learning Program
January, 2013

People matter, results count.

Objectives of Java-basic

- Purpose:
  - Basic understanding of basics of Java programming language

- Product:
  - To understand what is – Identifier, reserve keywords, data type, operators, type casting in Java
  - Understand – what is variable, constants, assignment etc.
  - Understand various flow control and looping mechanism in java, like – if, else, switch, while, do-while and for
  - Understand what is Array

- Process:
  - Theory Sessions along with relevant assignments
  - A review at the end of the session and a Quiz.
  - .

Topics

- Following topics will be covered
  - Language Basics
    - Identifier
    - Reserved Keywords
    - Data Types
    - Variable
    - Constants
    - Operators
    - Type Conversion
    - Assignment
    - Control Statements
    - Array

Java Reserved words

abstract
assert
boolean
break
byte
case
catch
char
class
const
continue
default
do

double
else
enum
extends
final
finally
float
for
goto
if
implements
import
instanceof

int
interface
long
native
new
package
private
protected
public
return
short
static

strictfp
super
switch
synchronized
this
throw
throws
transient
try
void
volatile
While

Data type

- Java is a strongly typed language.

  - 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.
- Java defines eight simple (or elemental) 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 wholevalued**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.

Data types - Integer

- Boolean: This group includes boolean, which is a special type for representing true/false values.
- **Integers**

  - Java defines four integer types: **byte, short, int, and long. All of these are signed, positive** and negative values.
- The Name and Width of these integer types vary widely, as shown below:

Name	**Width**
long	**64**
int	**32**
short	**16**
byte	**8**

Floating-Point Types

- Floating-point numbers, also known as *real numbers, are used when evaluating* expressions that require fractional precision.

  For example, calculations such as square root.

- There are two kinds of floating-point types, **float** and **double.**

Name	**Width**
double	64
float	32

Character Data Type

- In Java, the data type used to store characters is char.
- char letter = 'A'; (ASCII)
- char numChar = '4'; (ASCII)
- char letter = '\u000A'; (Unicode)
- char letter = '\u0004'; (Unicode)
- **Special characters**
- char tab = ‘\t’;

Boolean Type and Operators

Java has a simple type, called boolean, for logical values. It can have only one of two possible values, true or false.

boolean lightsOn = true;

boolean lightsOn = false;

&& (and)	(1 < x) && (x < 100)
|| (or)	(lightsOn) || (isDayTime)
! (not)	!(isStopped)

Primitive data types

data type	**default values**
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d.
char	null character, that is, '\u0000'.
boolean	false.
all reference types	null.

Variables

The variable is the basic unit of storage in a Java program. A variable is defined by the combination of an identifier, a type, and an optional initialize.

Declaring a variables

```
int a, b, c;                // declares three ints, a, b, and c.
int d = 3, e, f = 5;        // declares three more ints, initializing d and f.
byte z = 22;                // initializes z.
double pi = 3.14159;        // declares an approximation of pi.
char x = 'x';               // the variable x has the value 'x'.
```

Dynamic Initization

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

For example:

```
// 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);
        }
}
```

Constants

```
final datatype CONSTANTNAME = VALUE;
```

e.g.

```
final double PI = 3.14159;

final int SIZE = 3;
```

Operators

- Java provides a rich operator environment.

- Most of its operators can be divided into the following four groups:

- Type of operators
  - Arithmetic Operators
  - Bitwise Operators
  - Relational Operators and
  - Logical Operators.

- Java also defines some additional operators that handle certain special situations.

Arithmetic Operators

The following table lists the arithmetic operators:

Operator	**Result**
+	Addition
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

Increment and Decrement Operators

- The ++ and the – – are Java’s increment and decrement operators.
- The increment operator increases its operand by one. The decrement operator decreases its operand by one. For example, this statement:
- x = 42;
- y = ++x;
- In this case, y is set to 43 as you would expect, because the increment occurs *before x is* assigned to y
- y = 1 + x--;
- y = 1 + --x;

Bitwise operators

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

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assign

Relational operators

The *relational operators determine the relationship that one operand has to the other.*

Specifically, they determine equality and ordering. The relational operators are shown here:

Operator	**Result**
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Boolean Logical Operators

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
!=	Not equal to
?:	Ternary if-then-else

Shortcut Operators

Operator	*Example*	*Equivalent*
+=	i+=8	i = i+8
-=	f-=8.0	f = f-8.0
*=	i*=8	i = i*8
/=	i/=8	i = i/8
%=	i%=8	i = i%8

Ternary Operator(?)

- Java includes a special *ternary (three-way) operator that can replace certain types of* if-then-else statements.
- The **? has this general form:**
- *expression1* ***? expression2 : expression3***

- int i, k;
- i = 10;
- k = i < 0 ? -i : i; // get absolute value of I

- i = -10;
- k = i < 0 ? -i : i; // get absolute value of I

Shift Operators

Operator	Use	Operation
>>	op1 >> op2	shift bits of op1 right by distance op2
<<	op1 << op2	shift bits of op1 left by distance op2
>>>	op1 >>> op2	shift bits of op1 right by distance op2 (unsigned)

Logical Operators

Operator	Use	Operation
&	op1 & op2	bitwise and
|	op1 | op2	bitwise or
^	op1 ^ op2	bitwise xor
~	~op2	bitwise complement

Numeric Type Conversion

- Consider the following statements:

- byte i = 100;
- long l = i*3+4;
- double d = i*3.1+l/2;

- int x = l; (Wrong)
- long l = x; (fine, implicit casting)

Assignment Operator

- The *assignment operator is the single equal sign,* ***=.***
- ***The assignment*** operator works in Java much as it does in any other computer language.
- It has this general form:
- *var = expression;*
- Here, the type of *var must be compatible with the type of expression*
- *Example;*
- int x, y, z;
- x = y = z = 100; // set x, y, and z to 100

Operator Precedence

- Casting
- ++, -- (Unary operators)
- *, /, %
- +, -
- <, <=, >, >=
- ==, !=;
- &&, &
- ||, |
- =, +=, -=, *=, /=, %=

Control Statements

- A programming language uses *control statements to cause the flow of execution* to advance and branch based on changes to the state of a program.
- Java's program control statements can be put into the following categories:
- Selection Statements
- Iteration Statements, and
- Jump Statements

Java’s Selection Statements

Java supports two selection statements: **if and switch.**

if statement Syntax:

if (*condition*) *statement1;*

else *statement2;*

Example:

```
  if (i > 0)  {
          System.out.println("i is an “ +
             “integer greater than 0”);
  }
```

If…else Statement

```
if (condition)  {
      statement(s)-for-the-true-case;
}  else {
  statement(s)-for-the-false-case;
}
```

Example:

```
   int a, b;
   If (a < b) {
                a = 0;
   } else  {
                b = 0;
}
```

switch Statement

```
switch(i) {
    case 0:
      System.out.println("i is zero.");
      break;
    case 1:
      System.out.println("i is one.");
      break;
    case 2:
      System.out.println("i is two.");
      break;
    default:
      System.out.println("i is greater than 3.");
}
```

Nested switch Statements

```
switch(count) {
  case 1:
        switch(target) { // nested switch
                case 0:
                        System.out.println("target is zero");
                        break;
                case 1: // no conflicts with outer switch
                        System.out.println("target is one");
                break;
        }
        break;
case 2: // ...
```

Iteration Statements

Java’s iteration statements are 3 types:

for, while, and do-while.

While:

general form:

```
        while(condition) {
                // body of loop
        }
```

Example:

```
        int a = 10, b = 20;
        while(a > b) {
                System.out.println("This will not be displayed");
        }
```

for Loop

General form of the **for statement:**

```
for(initialization; condition; iteration) {
        // body
}
```

Examples:

```
// here, n is declared inside of the for loop
for(int n=10; n>0; n--)
        System.out.println("tick " + n);
}
```

do...while Loop

General form is:

```
do {
        // body of loop
}  while (condition);
```

Example:

```
int n = 10;
do {
        System.out.println("tick " + n);
        n--;
} while(n > 0);
```

Jump Statements

Java supports three jump statements:

break and

continue,

These statements transfer control to another part of your program.

break

```
class BreakLoop {
        public static void main(String args[]) {
                for(int i=0; i<100; i++) {
                        if(i == 10) break; // terminate loop if i is 10
                        System.out.println("i: " + i);
                }
                System.out.println("Loop complete.");
        }
}
```

continue

```
class Continue {
        public static void main(String args[]) {
                for(int i=0; i<10; i++) {
                        System.out.print(i + " ");
                        if (i%2 == 0)
                                continue;
                        System.out.println("");
                }
        }
 }
```

Java Arrays:

Arrays topics to be covered:

- What is an array?
- Declaration of an array
- Instantiation of an array
- Accessing array element
- Array length
- Multi-dimensional array

Java Arrays:

What is an array?
Array is one of the most fundamental data structure of any programming language that represents a collection of the <u>same type</u> of data.

Declaration:

datatype[] arrayname; or datatype arrayname[]

Example: int[] myList;

Creation:

arrayName = new datatype[arraySize];

Example: myList = new int[10];

Array Instantiation

- After declaring, we must create the array and specify its length with a constructor statement.

- Definitions:

Instantiation - In Java, this means creation

Constructor -   In order to instantiate an object, we need to use a constructor for this. A constructor is a method that is called to create a certain object.

We will cover more about instantiating objects and constructors later.

Array Instantiation

To instantiate (or create) an array, write the new keyword, followed by the square brackets containing the number of elements you want the array to have.

- For example,

```
//declaration
int ages[];
//instantiate object
ages = new int[100];
```

or, can also be written as,

```
//declare and instantiate object
int ages[] = new int[100];
```

Sample Program

```
//creates an array of boolean variables

 boolean results[] = { true, false, true, false };

 //creates an array of 4 double variables initialized  {100, 90, 80, 75};
 double []grades = {100, 90, 80, 75};

 //creates an array of Strings with identifier days
 //initialized. This array contains 7 elements

 String days[] = { “Mon”, “Tue”, “Wed”, “Thu”,  “Fri”, “Sat”,“Sun”};
```

Accessing an Array Element

To access an array element, or a part of the array, you use a number called an index or a subscript.

● index number or subscript

– assigned to each member of the array, to allow the program to access an individual member of the array.

– begins with zero and progress sequentially by whole numbers to the end of the array.

– NOTE: Elements inside your array are from 0 to (sizeOfArray-1).

Array of Elements

For example, given the array we declared a while ago, we have

```
//assigns 10 to the first element in the array

ages[0] = 10;

//prints the last element in the array

System.out.print(ages[99]);
```

Accessing an Array Element

```
public class ArraySample{
        public static void main( String[] args ){
                int[] ages = new int[100];
                for( int i=0; i<100; i++ ){
                        System.out.print( ages[i] );
                }
        }
}
```

Array Length

In order to get the number of elements in an array, you can use the length field of an array.

The length field of an array returns the size of the array. It can be used by writing, arrayName.length

Example:

```
int[] ages = new int[100];
for( int i=0; i<ages.length; i++ ){
        System.out.print( ages[i] );
}
```

Multidimensional Arrays

- Multidimensional arrays are implemented as arrays of arrays.

- Multidimensional arrays are declared by appending the appropriate number of bracket pairs after the array name.

For example,

```
   // integer array 512 x 128 elements
int[][] twoD = new int[512][128];

   // character array 8 x 16 x 24
char[][][] threeD = new char[8][16][24];

   // String array 4 rows x 2 columns
String[][] dogs = {{ "terry", "brown" },{ "Kristin", "white" },{ "toby", "gray"}, };
```

Array Length

- In order to get the number of elements in an array, you can use the length field of an array.
- The length field of an array returns the size of the array. It can be used by writing, arrayName.length

Example:

```
int[] ages = new int[100];
 for( int i=0; i<ages.length; i++ ){
                System.out.print( ages[i] );
}
```

Recap (Keywords)

If-else

Array

Reserved keywords

for loop

Data type

Casting

switch

Identifier

Variables and constants

Operator and operands

Multidimensional Array

Thank You For Your Time

People matter, results count.

COLLABORATIVE BUSINESS EXPERIENCE®