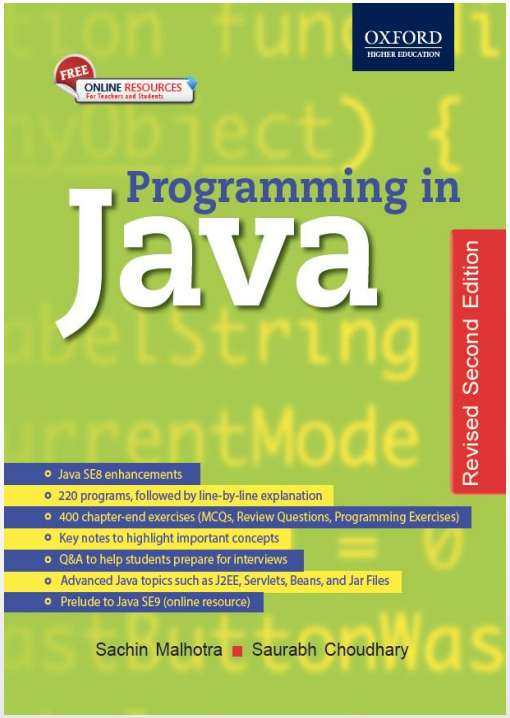
OXFORD
UNIVERSITY PRESS

Programming in Java

Revised 2nd Edition

Sachin Malhotra & Saurabh Choudhary



© Oxford University Press 2018. All rights reserved.

Chapter 3

Java Programming Constructs

© Oxford University Press 2018. All rights reserved.

Objective

- Understand how Variables are used in Java
- Understand the basic Data types
- Learn Expressions and Conditional Statements
- Understand all the available operations in Java
- Understand the basic of Conversion and Casting
- Understand Loops and Branching statements

© Oxford University Press 2018. All rights reserved.

Variables

- A symbolic name
- Used to store value which can change during execution of the program.
- int noOfWatts = 100; // variable declaration
- Declaration involves specifying the
  - type (Data Type),
  - name (Identifier) and
  - value (Literal) according to the type of the variable.

© Oxford University Press 2018. All rights reserved.

Primitive Data Types

- byte
- short
- int
- long
- float
- double
- boolean
- char

© Oxford University Press 2018. All rights reserved.

Default Values of Data Types and Their Range

Data Type	Default Value	Size	Range
byte	0	8	–128 to 127 (inclusive)
short	0	16	–32,768 to 32,767 (inclusive)
int	0	32	–2,147,483,648 to 2,147,483,647 (inclusive)
long	0L	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (inclusive)
float	0.0F	32	1.401298464324817e–45f to 3.402823476638528860e+38f
double	0.0D	64	4.94065645841246544e–324 to 1.79769313486231570e+308
char	'\u0000'	16	0 to 65535
boolean	false	Not defined	true or false

© Oxford University Press 2018. All rights reserved.

Identifiers

- Identifiers are names assigned to variables, constants, methods, classes, packages and interfaces.
- No limit has been specified for the length of a variable name.

© Oxford University Press 2018. All rights reserved.

Rules for Naming

- The first character of an identifier must be a *letter*, an *underscore* (_), or a *dollar sign* ($).
- Use *letter*, *underscore*, *dollar sign*, or *digit* for subsequent characters. Note that *white spaces* are not allowed within identifiers.
- Identifiers are *case-sensitive*. This means that *Total_Price* and total_price are different identifiers.
- Do not use Java's *reserved keywords*.

© Oxford University Press 2018. All rights reserved.

Valid and Invalid Identifiers

- *Legal Identifiers*
  - MyClass
  - $amount
  - _totalPay
  - total_Commission

- *Illegal Identifiers*
  - My Class
  - 23amount
  - -totalpay
  - total@commission

© Oxford University Press 2018. All rights reserved.

Naming Conventions

- **Class or Interface identifiers** begin with a capital letter. First letter of every internal word is capitalized. All other letters are in lowercase.
- **Variable or Method identifiers** start with a lowercase letter. First letter of every internal word is capitalized. All other letters are in lowercase.
- **Constant identifiers** All letters are specified in upper case. Underscore is used to separate internal words (_).
- **Package identifiers** consist of all lowercase letters.

© Oxford University Press 2018. All rights reserved.

Literals

- Value which can be passed to a variable
- Literals can be
  - numeric (for byte, short, int, long, float, double),
  - boolean,
  - character,
  - string or null literals.
- ***Integer literals*** can be decimal, octal, hexadecimal or even binary.
- All integer literals are of type int, by default.
- To define them as long, we can place a suffix of *L* or *l* after the number for instance:
  - long l = 2345678998L;

© Oxford University Press 2018. All rights reserved.

Octal, Binary and Hexa Literals

- Octal literals, the value must be prefixed with zero and only digits 0 through 7 are allowed.
  - int x = 011; //value in x is 9
- Hexadecimal literals are prefixed with *0x* or *oX*; digits 0 through 9

  and *a* through *f* (or *A* through *F*) are only allowed.
  - int y = 0x0001; //value in y is 1
- Binary literals are a combination of 0's and 1's. Binary literals can be assigned to variables in Java 7. Binary literals must be prefixed

  with 0b or 0B (zerob or zeroB)
  - char bin1 = 0b1010000; // value in bin1 will be P
  - char bin2 = 0b1010001; // value in bin2 will be Q
  - float bin3 = 0b1010000; // value in bin3 will be 80.0
  - int bin4 = 0b1010001; // value in bin4 will be 81

© Oxford University Press 2018. All rights reserved.

Underscores with Literals

- Java 7 onwards the readability of literals can be enhanced by using underscore with numeric literals.
- As the number of zeroes increase in a literal, counting the number of zeroes becomes tedious.
  - int numlit=100_000_000; // value in numlit will be 100000000
- Underscores can be used with decimal literals, hexa, binary, and octal literals.
  - int numlit=0x100_000; // value in numlit1 will be 1048576
  - int bin=0B1_000_000_000_000_001; // vale in bin will be 32769
  - float octlit=03_000; // value in octlit will be 1536.0
- float e = 4_.2_3f; // illegal use of underscore with a dot.

© Oxford University Press 2018. All rights reserved.

Valid and Invalid Use of Underscores within Literals

- **Note** Underscore can only be used with literal values.
- The following examples show some valid and invalid use of underscores.
  - int i =_23; // illegal, cannot start a literal with underscore
  - long f = 3_2_222_2_l; // invalid use of underscore between value and suffix
  - long f = 3_2_222_2l; // legal

© Oxford University Press 2018. All rights reserved.

Literals

- ***String literals*** consists of zero or more characters within double quotes.
  - String s = “This is a String Literal”;
- ***Null literal*** is assigned to object reference variables.
  - s =null;
- All ***floating literals*** are of type double, by default.
- For float literals, suffix *F* or *f*.
- for double literals, *D* or *d* are suffixed to the end (optional)
  - float f = 23.6F;
  - double d = 23.6;
- For ***char literals***, a single character is enclosed in single quotes.
  - char sample = ‘A’
  - the prefix \u followed by four hexadecimal digits represents 16-bit Unicode character:
  - char c = ‘\u004E’

© Oxford University Press 2018. All rights reserved.

Literals

Table 3.2 Unicode Escape Sequences to Represent Printable and Unprintable Characters

'\u0041'	Capital letter A
'\u0030'	Digit 0
'\u0022'	Double quote "
'\u003b'	Punctuation ;
'\u0020'	Space
'\u0009'	Horizontal Tab

Table 3.3 Special Escape Sequences

\\	Backslash
\"	Double quote
\'	Single quote
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab

© Oxford University Press 2018. All rights reserved.

Escape Sequence

Table 3.2 Unicode Escape Sequences to Represent Printable and Unprintable Characters

'\u0041'	Capital letter A
'\u0030'	Digit 0
'\u0022'	Double quote "
'\u003b'	Punctuation ;
'\u0020'	Space
'\u0009'	Horizontal Tab

Table 3.3 Special Escape Sequences

\\	Backslash
\"	Double quote
\'	Single quote
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab

© Oxford University Press 2018. All rights reserved.

Keywords

abstract	assert	boolean	break	byte
case	catch	char	class	continue
default	do	double	else	enum
extends	final	finally	float	for
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	const*	goto*
*const and goto are reserved keywords.				

© Oxford University Press 2018. All rights reserved.

Operators

- Assignment Operators (=)
- Arithmetic Operator (+, -, *, /, %)
- Relational Operators (==, <,>,<=,>=, !=)
- Boolean Logical Operators
  - Can be applied to Boolean operands (or expressions) and return a boolean value.
  - conditional OR (||) and conditional AND (&&)
  - logical OR (|),
  - logical AND (&),
  - logical XOR (^)
  - unary logical NOT (!)

© Oxford University Press 2018. All rights reserved.

Boolean Logical Operators

```
boolean a=true,b=false;
System.out.println(“Logical OR: “+ a +” | “+b+”: “+(a|b));
System.out.println(“Logical XOR: “+ a +” ^ “+b+”: “+(a^b));
System.out.println(“Logical AND: “+ a +” & “+b+”: “+(a&b));
System.out.println(“Logical NOT: !a : “+(!a));
System.out.println(“Conditional OR: “+ a +” || “+b+”: “+(a||b));
Syste m.out.println(“Conditional AND: “+ a +” && “+b+”: “+(a&&b));
// shortcut operator
a |= b;
System.out.println(“shortcut OR: “+ a +” | “+b+” = “+(a));
```

© Oxford University Press 2018. All rights reserved.

The Output

Logical OR: true | false: true

Logical XOR: true ^ false: true

Logical AND: true & false: false

Logical NOT: !a : false

Conditional OR: true || false: true

Conditional AND: true && false: false

© Oxford University Press 2018. All rights reserved.

Bitwise Operators

- Include and, or, xor, not, right shift, left shift and unsigned right shift.
- Operate on int and long values. If any of the operands is shorter than an int, it is automatically promoted to int before the operations are performed.

a & b	1 if both bits are 1
a | b	1 if either of the bits is 1
a ^ b	1 if both bits are different
~a	Complement the bits.
a << b	Shift the bits left by b positions. Zero bits are added from the LSB side. Bits are discarded from the MSB side.
a >> b	Shift the bits right by b positions. Sign bits are copied from the MSB side. Bits discarded from the LSB side.
a >>> b	Shift the bits right by b positions. Zero bits are added from the MSB side. Bits are discarded from the LSB side.

© Oxford University Press 2018. All rights reserved.

Bitwise Operators

```
class BitwiseOperatorDemo    {

public static void main(String args[])    {

int x=2,y=3;

System.out.println(“Bitwise AND: “+x+”&”+y+”=”+(x&y));

System.out.println(“Bitwise OR : “+x+”|”+y+”=”+(x|y));

System.out.println(“Bitwise XOR: “+x+”^”+y+”=”+(x^y));

System.out.println(“Bitwise NOT: ~”+x+”=”+(~x));

}}
```

© Oxford University Press 2018. All rights reserved.

The Output

- Bitwise AND: 2&3=2
- Bitwise OR : 2|3=3
- Bitwise XOR: 2^3=1
- Bitwise NOT: ~2=-3

© Oxford University Press 2018. All rights reserved.

Compound Assignment Operators

- +=
- -=
- *=
- /=
- %=
- &=
- |=
- ^=
- <<=
- >>=
- >>>=

- Note: The compound Assignment Operator includes an implicit cast.

© Oxford University Press 2018. All rights reserved.

Shift Operators

- Left shift
  - shifts the numbers of bits specified, towards left.
  - Bits are discarded from the left and added from the right with the value of bits being zero.
- Right shift
  - shifts the numbers of bits specified, towards right.
  - Bits are discarded from the right and added from the left side with the value of bits being that of the sign bit.
- Unsigned right shift,
  - shifts the numbers of bits specified, towards right.
  - Bits are discarded from the right and added from the left side with the value of bits being zero.

© Oxford University Press 2018. All rights reserved.

Shift Operators

- Assume x=4 and this x is to be shifted by the shift distance of 1.
- int y = x>>1;
  - y has the value 2, value is halved in each successive right shift

    00000000 00000000 00000000 00000100>>1

    00000000 00000000 00000000 00000010 (which is 2).

    int y = x<<1;
  - y has the value 8, value is doubled in each successive left shift
- int y = x>>>1;
  - same as right shift for +ve numbers.

© Oxford University Press 2018. All rights reserved.

Increment and Decrement

- ***Increment and Decrement Operators***
  - can be applied to all integers and floating-point types.
  - can be either in prefix (--x, ++x) or postfix (x--, x++) mode.

- **Prefix Increment/Decrement Operation**
  - int x = 2;
  - int y = ++x; // x = 3, y = 3
  - int z= --x; // x = 1, z = 1

- **Postfix Increment/Decrement Operation**
  - int x = 2;
  - int y = x++; // x == 3, y == 2
  - int z = x--; // x = 1, z = 2

© Oxford University Press 2018. All rights reserved.

Precedence and Associativity

Operators	Associativity
., [], (args), i++, i-¬	LR
++i, --i, +i, -i, ~, !	RL
new, (type)	RL
*, /, %	LR
+, ¬	LR
<<, >>, >>>	LR
<, >, <=, >=, instanceof	Non Associative
= =, ! =	LR
&	LR
^	LR
|	LR
&&	LR
| |	LR
? :	RL
=, +=, -=, *=, /=, %=, <<=, >>=, >>>=, &=, ^=, |=	RL

© Oxford University Press 2018. All rights reserved.

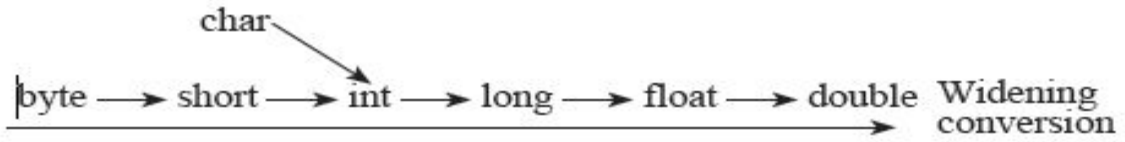
Your Turn

- What is the role of precedence and associativity?
- How are hexadecimal integers literals defined in java?
- How are binary literals defined in java?
- What are shift operators?
- Why and how underscores are defined in literals?

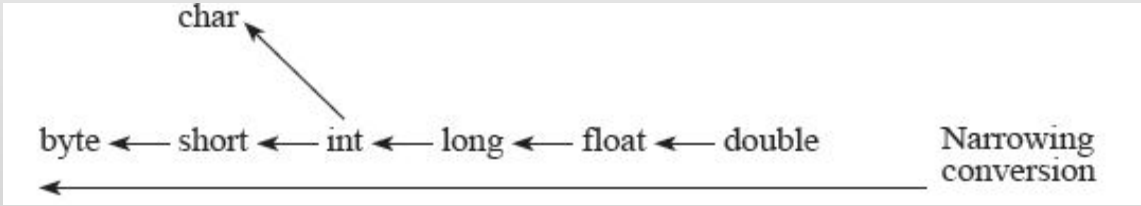
© Oxford University Press 2018. All rights reserved.

Conversion and Casting

- Conversions are performed automatically
  - For example a smaller box can be placed in a bigger box and so on.



- **Casting** also known as narrowing conversion (reverse of widening conversion).
  - A bigger box has to be placed in a small box.
  - Casting is not implicit in nature.
  - Use casting operator i.e. ()
  - int i = (int)(8.0/3.0);



© Oxford University Press 2018. All rights reserved.

Conversion and Casting

```
class CastingAndConversionExample
{  public static void main(String args[])
   {
            int i=(int)(8.0/3.0);
            int j=(int)2147483648.0f;
            System.out.println(“i = “+i+” j= “+j);
            byte b=(byte)257;
            short s=(short)65537;
            System.out.println(“b= “+b+” s= “+s);
            System.out.println(“ Converting int to char “ +(char)75);
            double d = i * b * s * 2.0;
            System.out.println(“Conversion to double result is : ”+d);
            i=b<<2;
            System.out.println(“i = “+i);
    }}
```

© Oxford University Press 2018. All rights reserved.

The Output

i = 2 j= 2147483647

b= 1 s= 1

Converting int to char K

Conversion to double result is : 4.0

i = 4

© Oxford University Press 2018. All rights reserved.

Flow of Control

- Conditional Statements
- Loops
- Branching statements
- Exceptions (chap 9)

© Oxford University Press 2018. All rights reserved.

Conditional Statement

if…else

The syntax of if statement is as follows:

```
if (x = = 0)
{// Lines of code}
else if(x = =1)
{// Lines of code}
……..
else
{// Lines of code}
```

© Oxford University Press 2018. All rights reserved.

Switch Case

```
switch (x) {
case 0:
    // Lines of code
    doSomething0();
    break;
case 1:
    // Lines of code
    doSomething1();
    break;
.
.
case n:
    // Lines of code
    doSomethingN();
    break;
default:
    doSomethingElse();
}
```

© Oxford University Press 2018. All rights reserved.

Switch Case

- x must be one of these: int, byte, short, char, enum, String (Java 7 onwards) or one of the wrapper class (namely: Byte for byte, Short for short, Character for char, Integer for int.).
- Can also be an expression that returns an int, byte, short or char.

© Oxford University Press 2018. All rights reserved.

For Loop

```
class ForDemo

{

public static void main(String args[])

{

for(int i=1;i<=5;i++)

System.out.println("Square of "+i+" is "+ (i*i));

}

}
```

© Oxford University Press 2018. All rights reserved.

The Output

The output is as follows:

Square of 1 is 1

Square of 2 is 4

Square of 3 is 9

Square of 4 is 16

Square of 5 is 25

© Oxford University Press 2018. All rights reserved.

while Loop

```
class WhileDemo
{
public static void main(String args[])
{
   int i=1;
   while(i<=5)
   {
        System.out.println("Square of "+i+" is "+ (i*i));
        i++;
   }    }
}
```

© Oxford University Press 2018. All rights reserved.

The Output

The output is as follows:

Square of 1 is 1

Square of 2 is 4

Square of 3 is 9

Square of 4 is 16

Square of 5 is 25

© Oxford University Press 2018. All rights reserved.

do…while Loop

```
class DoWhileDemo
{
   public static void main(String args[])
   { int i=1;
   do
   {
        System.out.println("Square of "+i+" is "+ (i*i));
        i++;
   }while(i<=5);
}}
```

© Oxford University Press 2018. All rights reserved.

The Output

The output is as follows:

Square of 1 is 1

Square of 2 is 4

Square of 3 is 9

Square of 4 is 16

Square of 5 is 25

© Oxford University Press 2018. All rights reserved.

for…each

```
for (type var : arr) {
   // Statements to repeat
}
```

- To access each value successively in a collection of values (like array).
- Like for loops, these loops perform a fixed number of iterations.
- But unlike them, for-each loop determines its number of steps from the size of the collection.

© Oxford University Press 2018. All rights reserved.

for…each

```
class PrimeDemo{
public static void main(String[] args){
int j,k;
System.out.print(“Prime numbers between 1 to 30 : “);
for (j = 1; j < 30; j++ ){
   for (k = 2; k < j; k++ ){
         if(j % k == 0) break;
   }
   if(j == k)
   {
   System.out.print(j+” “);
   }
}}}
```

© Oxford University Press 2018. All rights reserved.

break

```
class PrimeDemo{
public static void main(String[] args){
int j,k;
System.out.print(“Prime numbers between 1 to 30 : “);
for (j = 1; j < 30; j++ ){
   for (k = 2; k < j; k++ ){
         if(j % k == 0) break;
   }
   if(j == k)
   {
   System.out.print(j+” “);
   }
}}}
```

© Oxford University Press 2018. All rights reserved.

The Output

Prime numbers between 1 to 30 : 2 3 5 7 11 13 17 19 23 29

© Oxford University Press 2018. All rights reserved.

Labeled break

```
class LabeledBreakDemo{
public static void main(String args[])
{
   Outer : for(int i=0; i < 4 ; i++){
   for(int j = 1; j < 4; j++) {
   System.out.println("i:" + i + " j:" + j);
   if(i == 2) break Outer;
}}}}
```

© Oxford University Press 2018. All rights reserved.

The Output

i:0 j:1

i:0 j:2

i:0 j:3

i:1 j:1

i:1 j:2

i:1 j:3

i:2 j:1

© Oxford University Press 2018. All rights reserved.

continue

```
while ( i < 5){
   //doSomething1;
   if(i<4) continue;
        //doSomething2;
}
```

© Oxford University Press 2018. All rights reserved.

Labeled continue

```
jmp0: while (i < 5){

for (int i=0; i<4; i++){

   if( i==2) continue jmp0;

        //dosomething;

}

}
```

© Oxford University Press 2018. All rights reserved.

Summary

- Variables hold data at memory locations allocated to them.
- There are eight basic data types in java, namely, byte, short, int, long, float, double, boolean and char.
- Java does not leave the size of data types to the machine and the compiler.
- All integer (byte, short, int, long) and floating-point types (float, double) are signed in java.
- Operators can be classified as arithmetic, relational, logical, assignment, increment and decrement, conditional, bit-wise, and shift operator.
- Expressions are formed with variables and operators.
- Operators in java have certain precedence and associativity rules that are followed while evaluating expressions.
- Automatic type conversion takes place according to set rules in expressions with mixed types.
- Explicit type conversion (aka Casting) is also possible in java.

© Oxford University Press 2018. All rights reserved.