WSA | Forward looking IT finishing school

Java Programming Language SE – 6

Module 4 : Expressions and Flow Control

ORACLE®

Certified Professional

Java SE 6 Programmer

Objectives

- Distinguish between instance and local variables
- Describe how to initialize instance variables
- Identify and correct a Possible reference before assignment compiler error
- Recognize, describe, and use Java software operators
- Distinguish between legal and illegal assignments of primitive types

Objectives

- Identify boolean expressions and their requirements in control constructs
- Recognize assignment compatibility and required casts in fundamental types
- Use if, switch, for, while, and do constructions and the labelled

  forms of break and continue as flow control structures in a program

Relevance

- What types of variables are useful to programmers?
- Can multiple classes have variables with the same name and, if so, what is their scope?
- What types of control structures are used in other languages? What methods do these languages use to control flow?

Variables and Scope

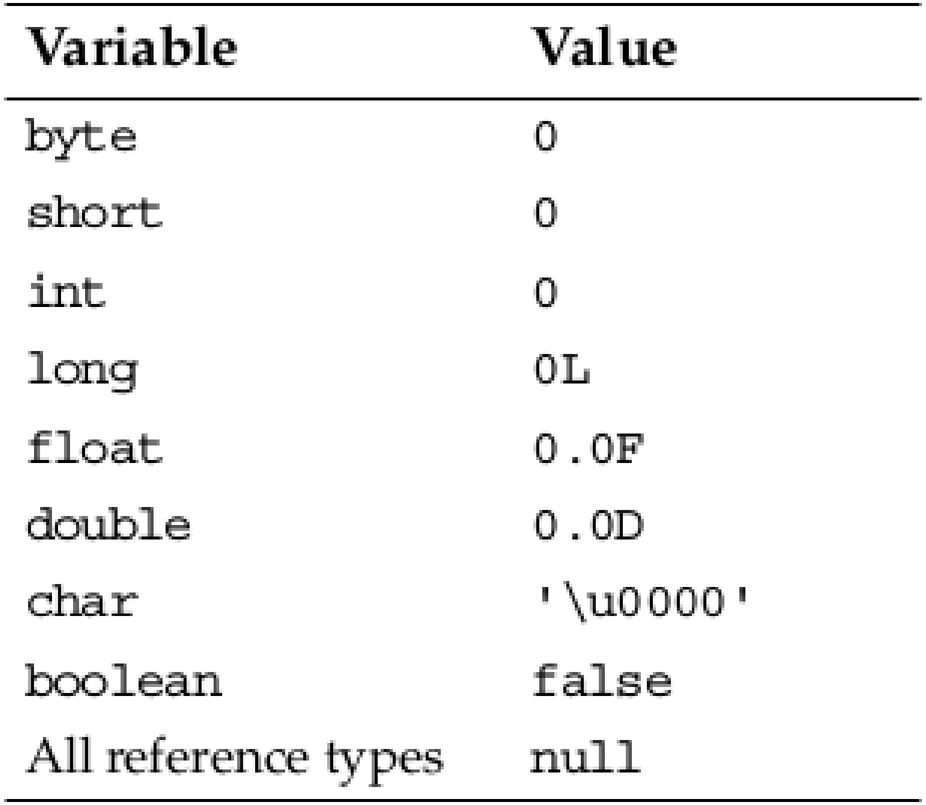
Local variables are:

- Variables that are defined inside a method and are called local, automatic, temporary, or stack variables
- Variables that are created when the method is executed are destroyed when the method is exited

Variable initialization comprises the following:

- Local variables require explicit initialization.
- Instance variables are initialized automatically.

Variable Initialization



Variable	Value
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0D
char	'\u0000'
boolean	false
All reference types	null

Initialization Before Use Principle

The compiler will verify that local variables have been initialized before used.

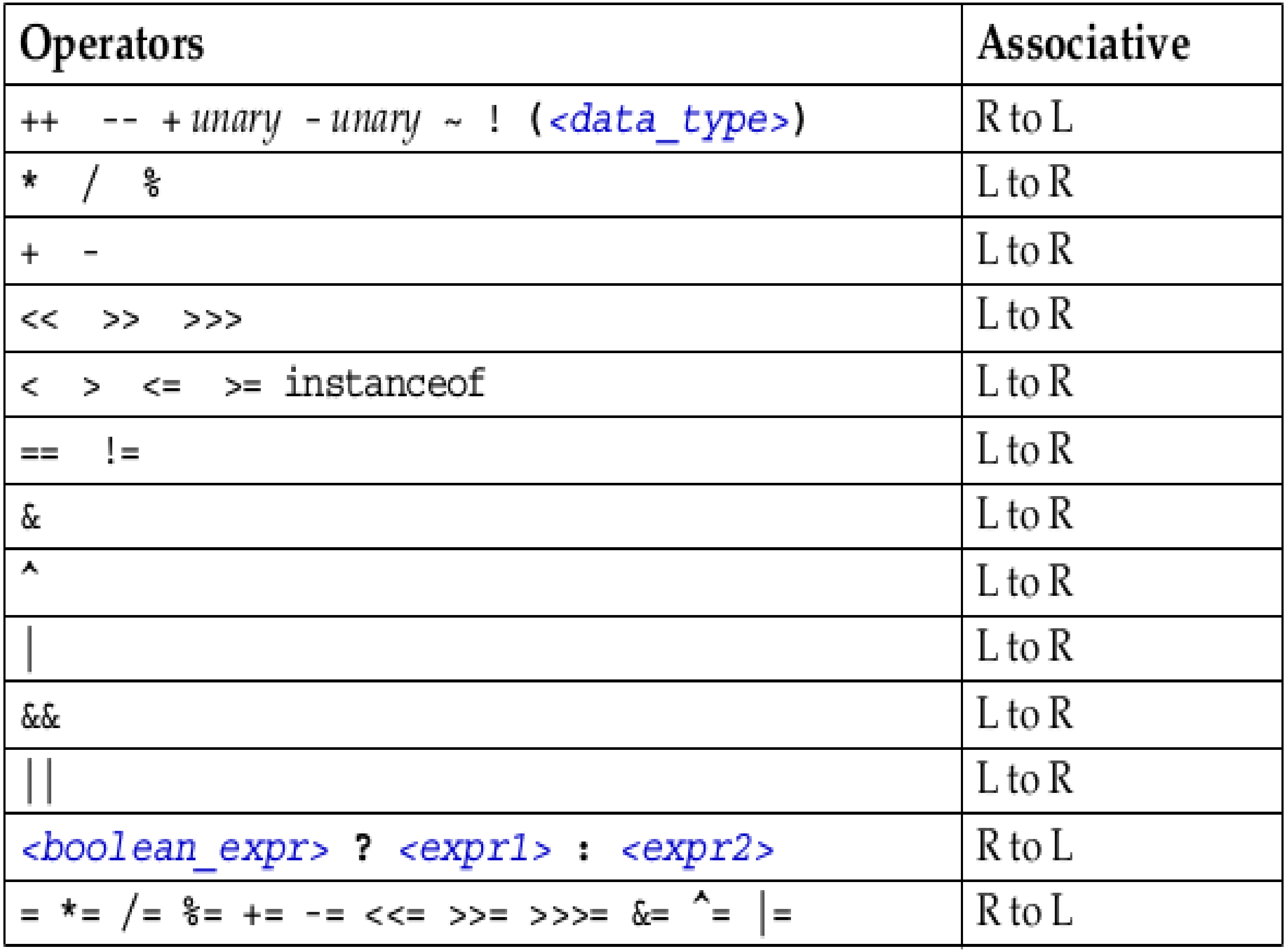
```
int x=8;

int y;

int z;

z=x+y;
```

Operator Precedence



Operators	Associative
++ -- +*unary* -*unary* ~ ! (<data_type>)	R to L
* / %	L to R
+ -	L to R
<< >> >>>	L to R
< > <= >= instanceof	L to R
== !=	L to R
&	L to R
^	L to R
|	L to R
&&	L to R
||	L to R
<boolean_expr> ? <expr1> : <expr2>	R to L
= *= /= %= += -= <<= >>= >>>= &= ^= |=	R to L

Logical Operators

- The boolean operators are:
  - ! – NOT
  - | – OR
  - & – AND
  - ^ – XOR
- The short-circuit boolean operators are:
  - && – AND
  - || – OR

Logical Operators

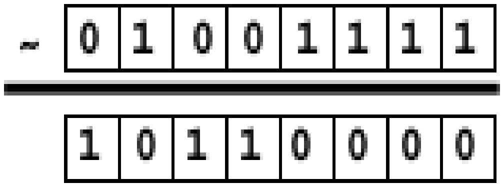
You can use these operators as follows:

```
MyDate d = reservation.getDepartureDate();
if ( (d != null) && (d.day > 31) {
// do something with d
}
```

Bitwise Logical Operators

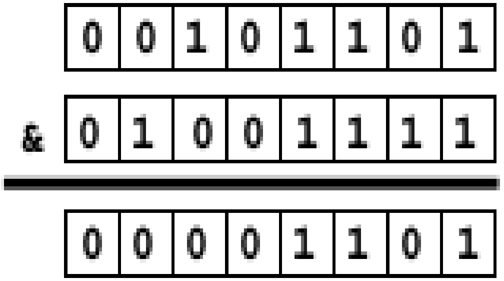
- The integer bitwise operators are:
  - ~ – Complement
  - ^ – XOR
  - & – AND
  - | – OR

Bitwise Logical Operators: Example



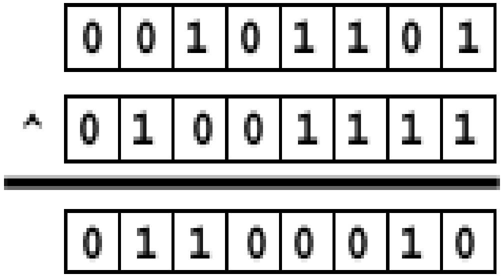
```
~ 0 1 0 0 1 1 1 1

  1 0 1 1 0 0 0 0
```



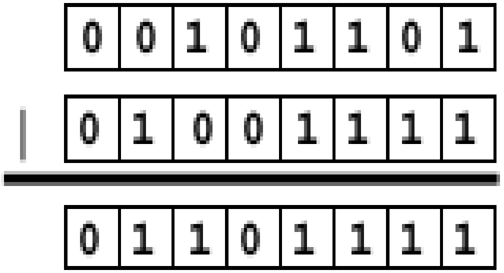
```
  0 0 1 0 1 1 0 1
& 0 1 0 0 1 1 1 1

  0 0 0 0 1 1 0 1
```



```
  0 0 1 0 1 1 0 1
^ 0 1 0 0 1 1 1 1

  0 1 1 0 0 0 1 0
```



```
  0 0 1 0 1 1 0 1
| 0 1 0 0 1 1 1 1

  0 1 1 0 1 1 1 1
```

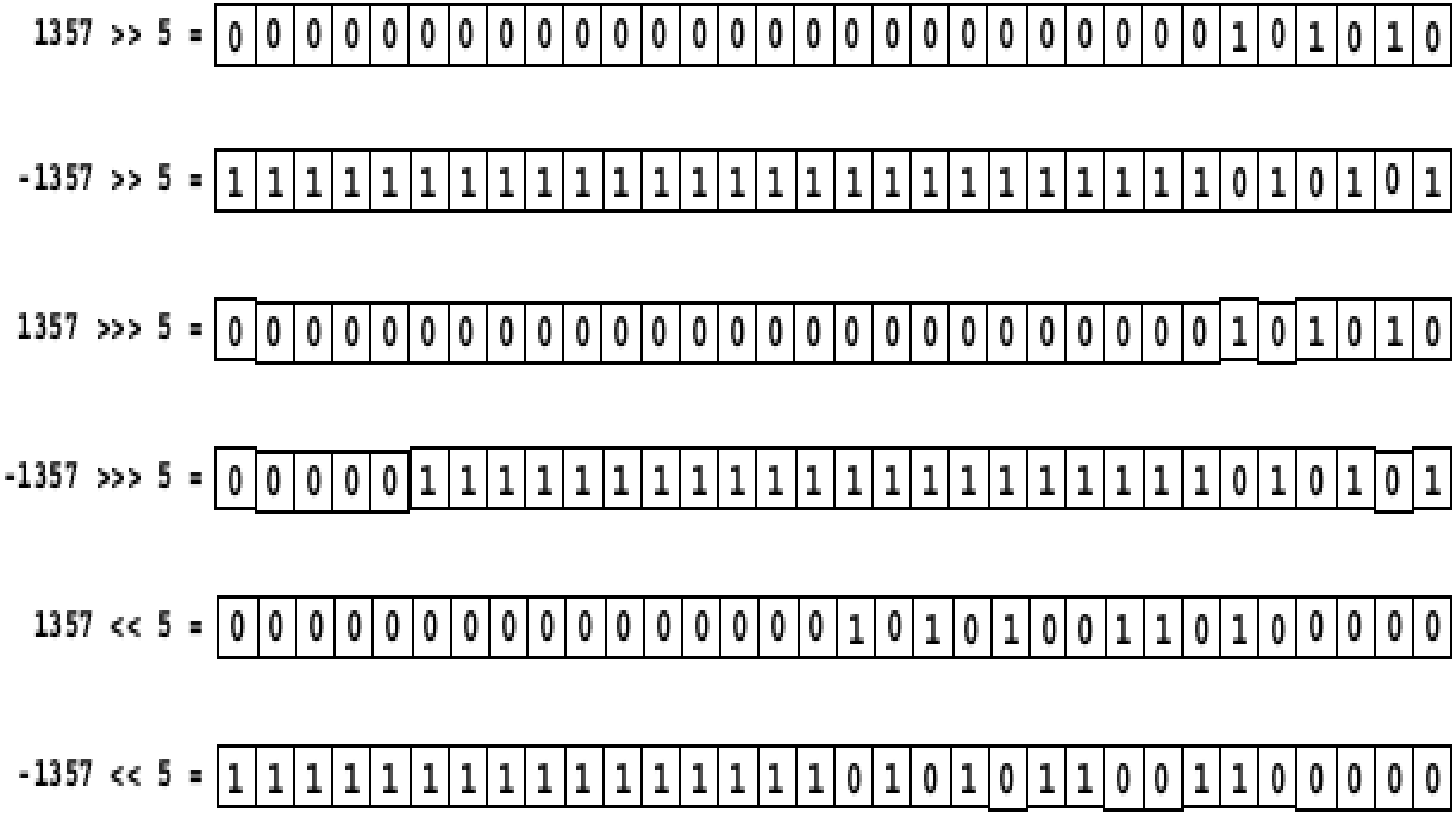
Right-Shift Operators >> and >>>

- Arithmetic or signed right shift (>>) operator:
- Examples are:
  - 128 >> 1 returns 128/2 1 = 64
  - 256 >> 4 returns 256/2 4 = 16
  - -256 >> 4 returns -256/2 4 = -16
- The sign bit is copied during the shift.
- Logical or unsigned right-shift (>>>) operator:
  - This operator is used for bit patterns.
  - The sign bit is not copied during the shift.

Left-Shift Operator <<

- Left-shift (<<) operator works as follows:
  - 128 << 1 returns 128 * 2 1 = 256
  - 16 << 2 returns 16 * 2 2 = 64

Shift Operator Examples



1357 >> 5 =	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0
-1357 >> 5 =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1
1357 >>> 5 =	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0
-1357 >>> 5 =	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1
1357 << 5 =	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	1	1	0	1	0	0	0	0	0
-1357 << 5 =	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	1	0	0	1	1	0	0	0	0	0

String Concatenation With +

- The + operator works as follows:
  - Performs String concatenation
  - Produces a new String:

```
String salutation = "Dr.";
String name = "Pete" + " " + "Seymour";
String title = salutation + " " + name;
```

Casting

- If information might be lost in an assignment, the programmer must confirm the assignment with a cast.
- The assignment between long and int requires an explicit cast.

```
long bigValue = 99L;
int squashed = bigValue;// Wrong, needs a cast
int squashed = (int) bigValue; // OK

int squashed = 99L;// Wrong, needs a cast
int squashed = (int) 99L;// OK, but...
int squashed = 99; // default integer literal
```

Promotion and Casting of Expressions

- Variables are promoted automatically to a longer form (such as int to long).
- Expression is assignment-compatible if the variable type is at least as large

```
long bigval = 6;// 6 is an int type, OK

int smallval = 99L; // 99L is a long, illegal

double z = 12.414F;// 12.414F is float, OK

float z1 = 12.414; // 12.414 is double, illegal
```

Simple if, else Statements

- The if statement syntax:

```
if ( <boolean_expression> )
<statement_or_block>
```

- Example:

```
if ( x < 10 )
System.out.println("Are you finished yet?");
```

or (recommended):

```
if ( x < 10 ) {
System.out.println("Are you finished yet?");
}
```

Complex if, else Statements

- The if-else statement syntax:

```
if ( <boolean_expression> )
<statement_or_block>
else
<statement_or_block>
```

- Example:

```
if ( x < 10 ) {
System.out.println("Are you finished yet?");
} else {
System.out.println("Keep working...");
}
```

Complex if, else Statements

- The if-else-if statement syntax:

```
if ( <boolean_expression> )
<statement_or_block>
else if ( <boolean_expression> )
<statement_or_block>
```

if-else-if statement: Example

- Example:

```
int count = getCount(); // a method defined in the class
if (count < 0) {
System.out.println("Error: count value is negative.");
} else if (count > getMaxCount()) {
System.out.println("Error: count value is too big.");
} else {
System.out.println("There will be " + count +
" people for lunch today.");
}
```

Switch Statements

The switch statement syntax:

```
switch ( <expression> ) {
case <constant1>:
<statement_or_block>*
[break;]
case <constant2>:
<statement_or_block>*
[break;]
default:
<statement_or_block>*
[break;]
}
```

Switch Statement Example

```
String carModel = ”STANDARD”;
switch ( carModel ) {
case DELUXE:
System.out.println(“DELUXE”);
break;
case STANDARD:
System.out.println(“Standard”);
break;
default:
System.out.println(“Default”);
}
```

Switch Statements

- Without the break statements, the execution falls through each subsequent case clause.

For Loop

- The for loop syntax:

  for (<init_expr>; <test_expr>; <alter_expr>)

  <statement_or_block>

For Loop Example

```
for ( int i = 0; i < 10; i++ )
System.out.println(i + " squared is " + (i*i));
```

or (recommended):

```
for ( int i = 0; i < 10; i++ ) {
System.out.println(i + " squared is " + (i*i));
}
```

While Loop

The while loop syntax:

```
while ( <test_expr> )
<statement_or_block>
```

While Loop Example

Example:

```
int i = 0;
while ( i < 10 ) {
System.out.println(i + " squared is " + (i*i));
i++;
}
```

The do/while Loop

- The do/while loop syntax:

```
do
<statement_or_block>
while ( <test_expr> );
```

The do/while Loop: Example

- Example:

```
int i = 0;
do {
System.out.println(i + " squared is " + (i*i));
i++;
} while ( i < 10 );
```

Special Loop Flow Control

- The break [<label>]; command
- The continue [<label>]; command
- The <label> : <statement> command, where <statement> should be a loop

The break Statement

```
do {
statement;
if ( condition ) {
break;
}
statement;
} while ( test_expr );
```

The continue Statement

```
do {
statement;
if ( condition ) {
continue;
}
statement;
} while ( test_expr );
```

Using break Statements with Labels

```
outer:
do {
statement1;
do {
statement2;
if ( condition ) {
break outer;
}
statement3;
} while ( test_expr );
statement4;
} while ( test_expr );
```

Using continue Statements with Labels

```
test:
do {
statement1;
do {
statement2;
if ( condition ) {
continue test;
}
statement3;
} while ( test_expr );
statement4;
} while ( test_expr );
```