Expressions and Flow Control

Mārtiņš Leitass

lattelecom

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 Scope Example

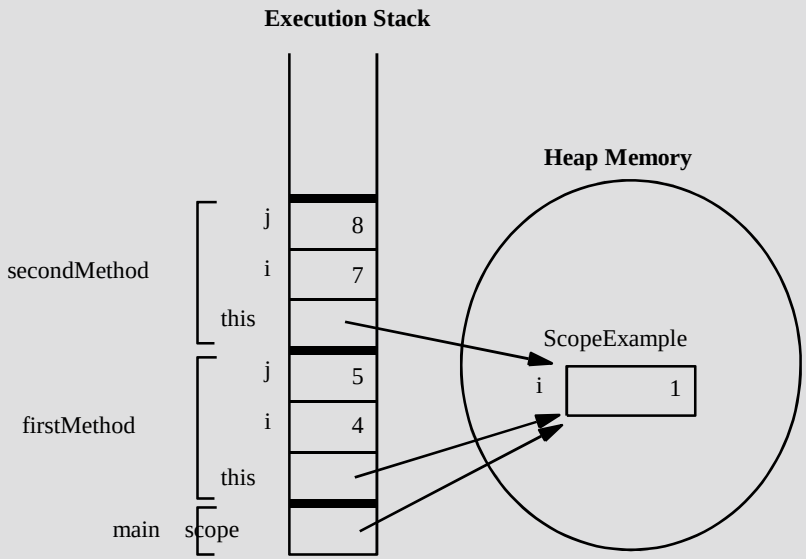
```
public class ScopeExample {
   private int i=1;

   public void firstMethod() {
      int i=4, j=5;

      this.i = i + j;
      secondMethod(7);
   }
   public void secondMethod(int i) {
      int j=8;
      this.i = i + j;
   }
}


public class TestScoping {
   public static void main(String[] args) {
      ScopeExample scope = new ScopeExample();

      scope.firstMethod();
   }
}
```



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.

```
3       public void doComputation() {
4           int x = (int)(Math.random() * 100);
5           int y;
6           int z;
7           if (x > 50) {
8              y = 9;
9           }
10          z = y + x;      // Possible use before initialization
11      }
```

javac TestInitBeforeUse.java

```
TestInitBeforeUse.java:10: variable y might not have been initialized
      z = y + x;    // Possible use before initialization
            ^
1 error
```

Arithmetic operators

- + additive operator (also used for String concatenation)
- - subtraction operator
- * multiplication operator
- / division operator
- % remainder operator

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;
```

- One argument must be a String object.
- Non-strings are converted to String objects automatically.

The Unary Operators

- The unary operators require only one operand; they perform various operations such as incrementing/decrementing a value by one, negating an expression, or inverting the value of a boolean
  - `+` Unary plus operator; indicates positive value (numbers are positive without this, however)
  - `-` Unary minus operator; negates an expression
  - `++` Increment operator; increments a value by 1
  - `--` Decrement operator; decrements a value by 1
  - `!` Logical complement operator; inverts the value of a boolean
  - `+=` concatenates or sums two operands

The Equality and Relational Operators

- == equal to
- != not equal to
- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to

Logical Operators

- The boolean operators are:

  ! – NOT & – AND
  | – OR ^ – XOR

- The short-circuit boolean operators are:

  && – AND || – OR

- You can use these operators as follows:

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

Precedence

- . [] {}
- ++ -- ! ~ instanceof
- new (type) expression
- * / %
- + -
- << >> >>>
- < > <= >=
- == !=
- &
- ^
- |
- &&
- ||

Casting

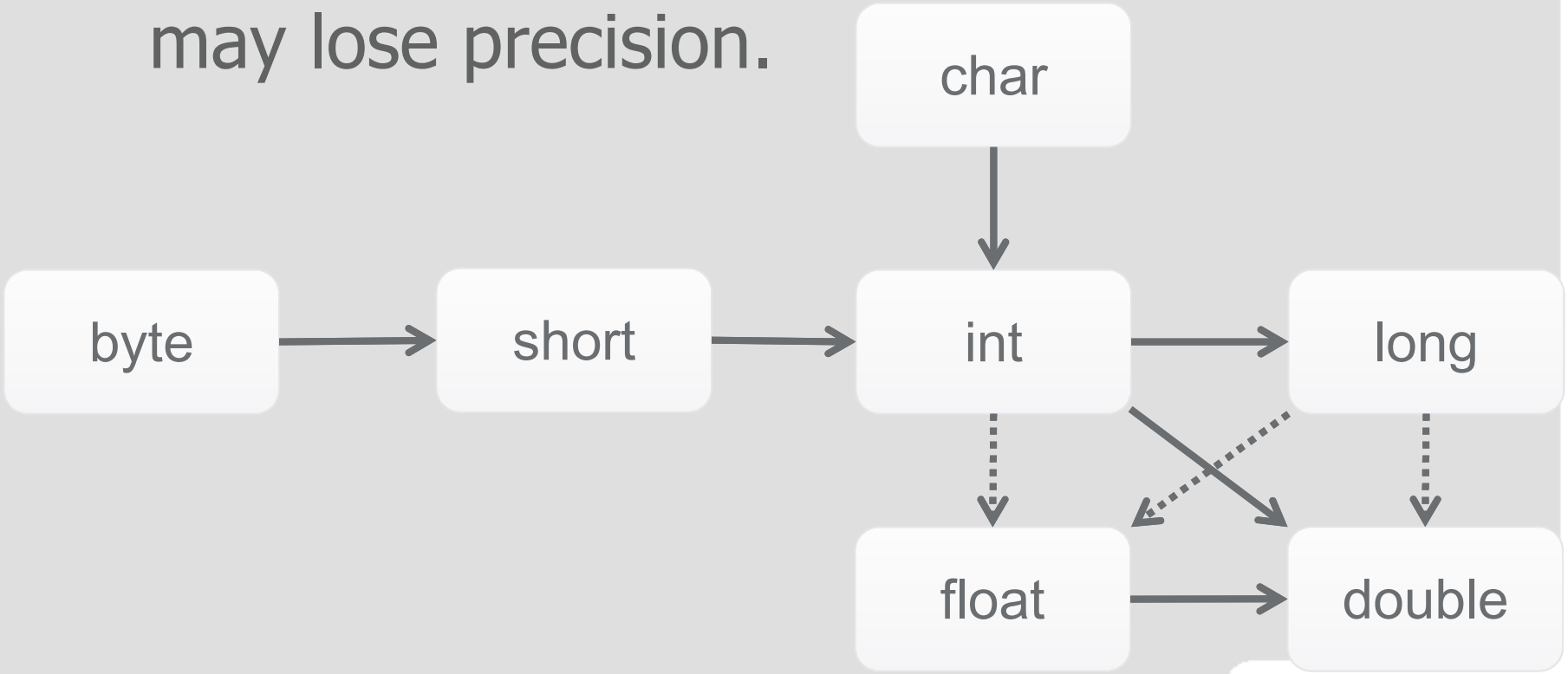
- If information might be lost in an assignment, the programmer must confirm the assignment with a cast.
- The assignment between long and int requires an explicit cast.

```
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
```

Legal conversions between primitive types

- Dotted arrows denote conversions that may lose precision.



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 (the same number of bits) as the expression type.

```
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>

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 Statements

A switch statement example:

```
switch ( carModel ) {
   case DELUXE:
      addAirConditioning();
      addRadio();
      addWheels();
      addEngine();
      break;
   case STANDARD:
      addRadio();
      addWheels();
      addEngine();
      break;
   default:
      addWheels();
      addEngine();
}
```

Switch Statements

This switch statement is equivalent to the previous example:

```
switch  (  carModel  )  {
    case  DELUXE:
        addAirConditioning();
    case  STANDARD:
        addRadio();
    default:
        addWheels();
        addEngine();
}
```

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

Looping Statements

The for loop:

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

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));
}
```

Looping Statements

The while loop:

while (*<test_expr>*)
** *<statement_or_block>***

Example:

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

Looping Statements

The do/while loop:

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

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

```
1    do  {
2        statement;
3        if  (  condition   )  {
4            break;
5        }
6        statement;
7    }  while  (  test_expr  );
```

The continue Statement

```
1    do {
2       statement;
3       if ( condition ) {
4          continue;
5       }
6       statement;
7    } while ( test_expr );
```

Using break Statements with Labels

```
1     outer:
2        do {
3           statement1;
4           do {
5              statement2;
6              if ( condition ) {
7                 break outer;
8              }
9              statement3;
10          } while ( test_expr );
11          statement4;
12       } while ( test_expr );
```

Using continue Statements with Labels

```
1     test:
2        do {
3           statement1;
4           do {
5              statement2;
6              if ( condition ) {
7                 continue test;
8              }
9              statement3;
10          } while ( test_expr );
11          statement4;
12       } while ( test_expr );
```