Module 4

Expressions and Flow Control

Objectives

- Distinguish between **instance** and **local variables**
- Describe variables **initialize** and **scope**
- Recognize, describe, and use Java **operators**
- Assignment compatibility and required casts in primitive types: **Promotion** and **Casting**
- Use `if`, `switch`, `for`, `while`, and `do`constructions and the labelled forms of `break` and `continue` as **flow control** structures in a program
- Input and output data using keyboard or GUI

Variables

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.

Instance 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`

Variable Scope Example

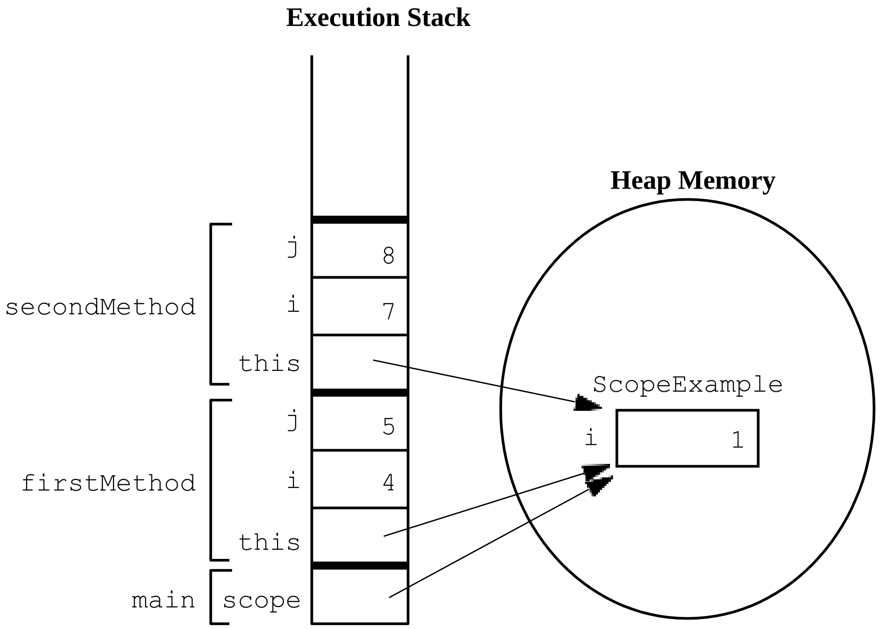
```
public class ScopeExample {
  private int i=1;

  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

    scope.firstMethod();
  }
}
```



Local Variables:
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
```

Operators and Expressions

Operators in Java programming language

Simple assignment operator: =

Arithmetic Operators: + - * / %

Unary Operators: + - ++ -- !

Relational Operators: == != > >= < <=

Logical Operators: && || & | ^

Conditional Operators: ? : (Ternary,shorthand for if-then-else)

Type Comparison Operator: instanceof

Bitwise and Bit Shift Operators: & | ~ ^ << >> >>>

- All binary operators except for the assignment operators are evaluated from left to right; assignment operators are evaluated right to left.
- Operator Precedence: postfix-unary-multiplicative-additive-shift-relational-instanceof-equality-&-^-|-&&-||-?:-assignment

Conditional operator

- <boolean_expr> ? <expr1> : <expr2>

```
int grade = 80;
String status =(grade >= 60) ? "Passed" : "Fail";
```

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
}
```

Bitwise Logical Operators

- The integer *bitwise* operators are:

```
~ – Complement  & – AND
^ – XOR         | – OR
```

- Byte-sized examples include:

```
                            0 0 1 0 1 1 0 1
~ 0 1 0 0 1 1 1 1         & 0 1 0 0 1 1 1 1
-----------------         -----------------
  1 0 1 1 0 0 0 0           0 0 0 0 1 1 0 1

  0 0 1 0 1 1 0 1           0 0 1 0 1 1 0 1
^ 0 1 0 0 1 1 1 1         | 0 1 0 0 1 1 1 1
-----------------         -----------------
  0 1 1 0 0 0 1 0           0 1 1 0 1 1 1 1
```

Right-Shift Operators >> and >>>

- *Arithmetic* or *signed* right shift (>>) operator:
  - Examples are:

```
128 >> 1 returns 128/21   =
                          64
256 >> 4 returns 256/24   =
```

  - 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 =`	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	0	1	1	0	1
`-1357 =`	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	1	0	0	1	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 =`	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

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

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 by invoke it's `toString()` method automatically.

Promotion and Type Casting

Promotion

- Automatic promotions:
  - If you assign a smaller type to a larger type
  - If you assign an integral type to a floating point type
- Examples of automatic promotions:

```
long big = 6;
```

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

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

Type Casting

- Syntax:

```
identifier = (target_type) value
```

- Example of potential issue:

```
int num1 = 53; // 32 bits of memory to hold the value
int num2 = 47; // 32 bits of memory to hold the value
byte num3; // 8 bits of memory reserved
num3 = (num1 + num2); // causes compiler error
```

- Example of potential solution:

```
int num1 = 53; // 32 bits of memory to hold the value
int num2 = 47; // 32 bits of memory to hold the value
byte num3; // 8 bits of memory reserved
num3 = (byte)(num1 + num2); // no data loss
```

Compiler Assumptions for Integral and Floating Point Data Types

- Example of potential problem:

```
short a, b, c;
a = 1 ;
b = 2 ;
c = a + b ; //compiler error
```

- Example of potential solutions:
  - Declare `c` as an `int` type in the original declaration:

```
int c;
```

  - Type cast the (`a+b`) result in the assignment line:

```
c = (short)(a+b);
```

Floating Point Data Types and Assignment

- Example of potential problem:

```
float float1 = 27.9;//compiler error
```

- Example of potential solutions:
  - The `F` notifies the compiler that 27.9 is a `float` value:

```
float float1 = 27.9F;
```

  - `27.9` is cast to a `float` type:

```
float float1 = (float) 27.9;
```

Flow Control

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("  you finished yet?");
  Are
```

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        yet?");
  finished
  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  too big.");
  is
  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.

switch Expression(Since Java 14)

```
int j = switch (s) {
        case "Foo" -> 1;
        case "Bar" -> 2;
        default -> {
            System.out.println("hmmm...");
            yield 0;
        }
    };
```

The for Statement

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));
}
```

for statement

- for(;;) { }
  - creates a loop that repeats forever.
- Comma Separator

```
int j, k;
for (j = 3, k = 6; j + k < 20; j++, k +=2)
   { System.out.println("j is " + j + " k is " +
   k);
```

for statement

- You cannot mix expressions with variable declarations, nor can you have multiple declarations of different types.
  - int i = 7;
    for (i++, int j = 0; i < 10; j++) { } // illegal!
  - for (int i = 7, long j = 0; i < 10; j++) { } // illegal!

Enhanced *for* Loops

- Since JDK 5
- Work more easily with arrays and collections.
- Eliminates the loop counter, syntax is:

  for (type variable_name : array | collection)

Enhanced *for* Loops

- perform identical processing on every element of an array：

```
float sumOfSquares(float[] floats)
  { float sum = 0;
  for (int i=0; i<floats.length; i++)
    sum += floats[i];
  return sum;
}
```

Enhanced *for* Loops

- With enhanced for loops, this method can be rewritten as:

```
float sumOfSquares(float[] floats) {
   float sum = 0;
   for (float f:floats)
      sum += f;
   return sum;
}
```

The while Statement

The `while` loop:

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

Example:

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

The do-while Statement

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

- **break** ***[<label>]***;
- **continue** ***[<label>]***;
- ***<label>*** **:** ***<statement>*** , where ***<statement>*** should be a loop
- **return** [<*expression*>] **;**
- The exit() method: **System.exit(0);**
- **NO** **goto** statement

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

Using continue Statements with Labels

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

Input and Output

Input and Output

- **Formatted output using printf() method**
- **Create a Java program that gets input from the keyboard**
  - **Use Scanner class**

```
 java.util.Scanner input = new java.util.Scanner( System.in );
input.nextLine();// read a line
input.nextInt();  //read an integer
nextFloat();nextDouble()…
hasNext();hasNextInt()…
```

  - **Use the BufferedReader class in the java.io package**

```
BufferedReader dataIn = new BufferedReader(
                                   new InputStreamReader(System.in) );
String temp = dataIn.readLine();
int val = Integer.parseInt(temp)
```

  - **Use the JOptionPane class in the javax.swing package**

printf()

- System.out.printf(format, args);
  - System.out.printf("姓名:%s, 成绩:%4.2f\n",
                    name , score)

Using Scanner class

- java.util.Scanner
  - String nextLine(): read a line
  - int nextInt()： read a int number
  - float nextFloat()、double nextDouble()…
  - hasNext()、hasNextInt()…

- Scanner sc=new Scanner(System.in);
  int i = sc.nextInt();

- TestScanner.java

Using *JOptionPane* Class

- javax.swing package
- JOptionPane makes it easy to pop up a standard dialog box that prompts users for a value or informs them of something.
  - String JOptionPane.showInputDialog(msg) ;
  - void JOptionPane.showMessageDialog(null,msg);

- Person.java

using *BufferedReader* Class

- Add this at the top of your code:

  import java.io.*;

- Add this statement:

  BufferedReader dataIn =
    new BufferedReader(new InputStreamReader(System.in));

- Declare a String variable to get the input, and invoke the readLine() method to get input from the keyboard.

  String temp = dataIn.readLine();

- Then parse to value if needed.

  int val = Integer.parseInt(temp);

- GetInputFromKeyboard.java

using *Console* Class

- java.io.Console
- get the system Console:
  - System.console()
- read a line from console:
  - console.readLine()
- read password from console:
  - console.readPassword()
- TestConsole.java

Summary

- Variables and Scope
- Operators and Expressions
- Promotion and Type Casting
- Flow Control
- Input and Output