Chapter 5

More on control structures

Lecture slides for:

Java Actually: A Comprehensive Primer in Programming

Khalid Azim Mughal, Torill Hamre, Rolf W. Rasmussen

Cengage Learning, 2008.

ISBN: 978-1-844480-933-2

`http://www.ii.uib.no/~khalid/jac/`

Permission is hereby granted to use these lecture slides in conjunction with the book.

Modified: 14/11/08

Overview

- Extended assignment operators
- Increment and decrement operators
- Counter-controlled loops: `for` loop
- Multiple-selection statement: `switch`

Extended assignment operators

- Common operation: compute the new value of a variable when its old value is an operand of a binary arithmetic operator.

```
double price = 10.50, shipping = 5.50;
price = price + shipping; (1)
```

- (1) can be written as the following statement which uses the += extended assignment operator:

```
price += shipping; (1')
```

- The expression on the right hand side in (1') can be an arbitrary arithmetic expression.

Extended assignment operator:	Meaning:
`x += (y)`	`x = x + (y)`
`x -= (y)`	`x = x - (y)`
`x *= (y)`	`x = x * (y)`
`x /= (y)`	`x = x / (y)`
`x %= (y)`	`x = x % (y)`

- See class `ExtendedAssignmentOperators` in Program 5.1, p. 100.

Unary increment and decrement operators

- Shorthand for increment and decrement by 1.
- Given that `x` has the value 4, the table below shows the value of the expression (that uses either an increment or an decrement operator) and the value of `x` after expression evaluation.

Operator:	Expression (x = 4)	Value of expression	Value of x after expression evaluation
pre-increment operator	`++x`	5	5
post-increment operator	`x++`	4	5
pre-decrement operator	`--x`	3	3
pre-decrement operator	`x--`	4	3

- Note the side effect on the value of the variable.
- See class `IncrAndDecrOperators` in Program 5.2, p. 101.

`for` loop

- The general loop construction is mostly used for *counter-controlled loops* where the number of *iterations* is known beforehand.

Syntax:

```
for (<initialization>; <loop condition>; <updating>) {
    /* loop body */
}
```

- <*initialization*>: expression for initializing the *loop variable.*
- <*loop condition*>: expression to determine if the loop body should be executed, and in which the loop variable occurs.
- <*updating*>: expression which modifies the loop variable.

Semantics:

<*initialization*>

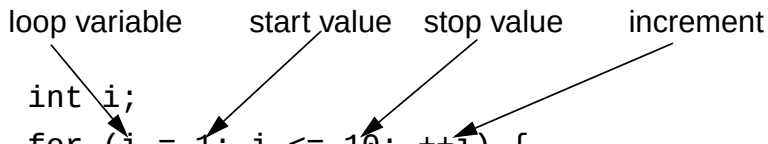
repeat while <*loop condition*> *is true:*

<*loop body*>

<*updating*>

Example: for loop

Print the square root of numbers from 1 to 10.




```
int i;
for (i = 1; i <= 10; ++i) {
    System.out.println(i + "\t\t" + Math.sqrt(i));
}
```

Scope of the loop variable in the `for` loop

- <*initialization*> and loop body can contain declarations of *local variables.*
- These variables can only be used in the `for` loop.
- Loop variable can be declared in the <*initialization*>:

```
// Print the square root of numbers from 1 to 10.
for (int i = 1; i <= 10; ++i) {
    System.out.println(i + "\t\t" + Math.sqrt(i));
}
// Local variables declared in the loop are not accessible
// after the loop finishes execution.
```


Writing a program

1. Problem definition (translating a telephone number to words)

Write a program that reads a phone number from the terminal window and prints the digits as words. When the user types:

55584152

the program prints in the terminal window:

```
five five five eight four one five two
```

2. Problem analysis and specification

- How should the telephone number be typed?

  Data format: `55-58-41-52` or `55584152`. (We choose the last format.)
- We read the number as a string, convert each character (which is a digit) to the corresponding word, building a result string with the words.

3. Design:

Algorithm	Data
1. *Initialize the variables* 2. *Read the telephone number as a string* 3. ***repeat while*** *not all the digits in the string have been translated:* *read a digit from the string* *translate the digit character to the corresponding word* *update the result string* 4. *Print the result string with the words.* *Step 3 comprises a loop.*	*A variable,* `inputPhoneString`*, to read the telephone number.* *A variable,* `aChar`*, to hold the current digit character.* *A variable,* `outputPhoneString`*, to store the words.*

4. Coding: see class `Telephone` **in Program 5.3, p. 104.**

- Exercise: modify the program so that the correct number of digits are read for the phone number.

More on the `for` loop

- The `for` loop can replace a `while` loop.

```
Scanner keyboard = new Scanner(System.in);

int num = keyboard.nextInt();
int sum = 0;
while (num != 0) {  // While loop
    sum += num;
    num = keyboard.nextInt();
}
```

(1) With the `for` loop:

```
int num = keyboard.nextInt();
int sum = 0;
for ( ; num != 0; ) {  // Both semicolons must be included.
    sum += num;
    num = keyboard.nextInt();
}
```

(2) Logical error:

```
int num = keyboard.nextInt();
for (int sum = 0; num != 0; sum += num) {
    num = keyboard.nextInt();
}
```

- The `for` loop cannot replace a `do-while` loop because the test comes after the loop body.
- If all the expression are left out in the `for` loop, we get an "infinite loop":

```
for (;;) { // The "crab" is always true.
    System.out.println("Stop! I want to get off!");
}
```

  – An infinite loop can be terminated by a `return` or a `break` statement in the loop body (explained later).

Nested loops

Example: Print the multiplication tables from 1 to 10.

```
for (int i = 1; i <= 10; ++i) {
    System.out.println("Multiplication table for " + i);
    for (int j = 1; j <= 10; ++j) {
        System.out.println(i + "\tx\t" + j + "\tis\t" + (i*j)); // (1)
    }
}
```

- How many times is (1) executed?

"Backwards" `for` loop

Example: Print the square root of numbers from 10 to 1.

```
for (int i = 10; i > 0; i--) {
    System.out.println(i + "\t\t" + Math.sqrt(i));
}
```

Examples with loops

- One-too-many and one-too-little errors:
  (The banana problem: "I know how to spell banana. I just don't know when to stop.")

```
result = 0;
for (int counter = 2; counter < numOfTimes; counter += 2) {
    result += counter;
} // if numOfTimes is an even number, its value will not be added.
```

- Correct initialization of the loop variable:

```
result = 0;
do {
    result *= 2;
} while (result < 100); // Infinite loop
```

- Check that the condition is correct:

```
// Example 1
result = 0;
for (int counter = 1; counter != 100; counter += 2) {
    result += counter;
} // Infinite loop
```



```
// Example 2
int sum = 0;
for (int i = 1; i < 1; i++) { // Test is false on entering the loop.
    sum += i;
}
```

- Loop invariants can be moved out of the loop body, called *loop optimization*.

```
// Alternative 1
double sum = 0.0, pi = 3.14;
for (int i = 0; i < 100; ++i) {
    sum += Math.pow(pi * 2.0, i);// (pi*2) is constant in the loop,
                                 // can be moved out.
}
// Alternative 2
double sum = 0.0, pi = 3.14;
double inv = pi * 2.0;
for (int i = 0; i < 100; ++i) { sum += Math.pow(inv, i); }
```

"Hand simulate" always the first and the last iteration to avoid one-too-many and one-too-little errors, and unintentional infinite loops. Check also some of the intermediate iterations.

Choosing the right loop

- `for` loop is usually used for mathematical computations, like adding finite series.
- `while` loop is usually used for situations where the loop condition can become true at any time, i.e. where the number of iterations is not known beforehand.
- `do-while` loop is used in the same situations as for the `while` loop, but when the loop body must be executed at least once.

Choice of loop

- *Bake a cake for each or your 6 friends.*
- *Stand at the end of a bowling lane and pick up all the bowling pins, one at a time, that have been fallen down.*
- *Search through a deck of cards for the Queen of Spade.*
- *Ask a question. Continue to ask if the answer is wrong.*

`break` statement in loops

- Execution of the `break` statement in the loop body transfers the program control out of the loop.
  - Execution continues after the loop.
- Typical use of the `break` statement is too early termination of the loop.

```
for (int i = 1; i <= 10; ++i) {
    if (i == 6) break; // Loop terminates when i is assigned the value 6
    System.out.println(i + "\t\t" + Math.sqrt(i));
}
// Continue here.
```

`continue` statement in loops

- Execution of a `continue` statements in the loop body results in the rest of loop body being skipped, and execution continues with the next iteration of the loop.
- In a `while` or `do-while` loop, the loop condition is tested immediately after the continue statement has executed.
- In a `for` loop, after the `continue` statement is executed, first the updating of the loop variable is done, and then the loop condition is tested.

```
for (int i = 1; i <= 10;  /* (1) */ ++i) {
    if (i == 6) continue; // Rest of the loop body is not executed
                          // when i gets the value 6. Control transferred to (1).
    System.out.println(i + "\t\t" + Math.sqrt(i));
}
```


More on selection: `switch` statement

- The `switch` statement can be used to select one of several actions, based on the value of an expression. (See Figure 5.4, p. 113).

Syntax:

```
switch (switch-expression) {
    case value1:
        // list of actions
        break;
    case value2:
        // list of actions
        break;
    ...
    case valueN:
        // list of actions
        break;
    default:
        // default list of actions
} // end switch
```

`switch` statement (cont.)

Semantics:

- Compare the `switch-expression` value with each `case` label one at a time.
  - if the `switch-expression` value is equal to the `case` label, execute the corresponding list of actions.
  - if no `case` label is equal to the `switch-expression` value, execute the list of actions for the `default` label.
- The `default` label is optional.
- If no `case` label is equal to the `switch-expression` value, and the `default` label is missing, no actions are executed, and program execution continues after the `switch` statement.
- The label values must be unique *literals* (integers, characters or enum constants).
- The `break` statement is used to mark the end of the list of actions (which can contain zero or more actions).
  - execution of the `break` statement in a `switch` statement transfers the program control to after the `switch` statement.
  - Without the `break` statement execution will continue in the subsequent list of actions.

- It is not necessary to enclose the list of actions in the block notation, `{ }`.
- The `case` labels and the `default` label can be specified in any order, as long as each list of actions is terminated by a `break` statement.
- The `switch-expression` value and the `case` label are compared for *equality*, while the condition in an `if-else` statement can be an *arbitrary boolean expression*.

Example using the `switch` *statement: class* `TelephoneII` *in Program 5.6, p. 114.*

- How is this program different from the one in Program 5.3, p. 104, which uses an `if-else` statement?

- *Fall through* in a `switch` statement is illustrated in Program 5.7, p. 116 (class `Seasons`).