Chapter 6

JC08-5

(A. Nguyen)

Iterations

JC 6.1-6.4

Iterations: Definition

- Repeat the same sequence of instructions multiple times; e.g., calculate the money amount by accumulating interest over many years
- There are **3 parts** to a loop:
  - Start with initial value(s) of variable(s)
  - Change the value(s) of the variable(s) in each cycle
  - Stop when the tested condition becomes false – The condition must become false some time, or else infinite loop
- Supported by high-level programming languages
- There are 3 kinds of loops in Java: `while, for, do-while`
- `while` may be written as `for`, and vice-versa

Iterations: “while” loop in Java

```
// set up the condition
while (<this condition holds>)
{
    ... // do something
    … // update something to change the condition
}
```

For example: calculate the sum of the squares of the 1st n integers

```
// int n was assigned to some positive value
int k=1;
int sum = 0;
while (k <= n)
 {
     sum += k * k;   //  add k * k to sum
     k++;            //  increment k by 1
}
```

```
sum = sum + k*k;
      k = k + 1;
```

Iterations: “for” loop in Java

```
for (<initial setup>; <as long as this condition holds>;
            <adjust variable(s) at the end of each iteration>)
{
   ... // do something
}
```

For example: calculate the sum of the squares of the 1st n integers

```
// int n was assigned to some positive value
int sum = 0;
for ( int k = 1;  k <= n;  k++)
{
    sum += k * k;   // add k * k to sum
}
```

Increment *k* by 1

"Initialization" of the `for` loop

The only difference between these 2 code fragments is the scope of counter `k`.

Counter inside for loop	Counter outside for loop

for (int k = 0; k < n ; k++)
{
    // code for loop body here
}
	int k = 0;
	for (; k < n ; k++)
	{
	    // code for loop body here
	}

`k` IS NOT "visible" after the `for` loop b/c scope is the `for` body	`k` IS "visible" after the `for` loop b/c the scope is outside & including the `for` body

Empty initialization – must still have semi-colon

"Change" of the `for` loop

The "change" may be calculated inside the loop body, depending on other info.

Counter inside for loop	Counter outside for loop

`for (int k = 0;  k < n ;  k++)`
`{`
`      // code for loop body here`
`}`
	`for (int k = 0;  k < n ; )`
	`{`
	`      // some code here`
	`      n -= x; // x is some var.`
	`}`

`k` is changed in the `for` header	`n` is changed in body

Empty "change" – must still have semi-colon

Characteristics of a `do-while` loop

- Composition `do-while` loop:

```
do { …
    change
    …
} while(condition);
```

- A `do-while` loop is always executed at least once, because the condition is first tested after the first iteration. Thus, a `do-while` loop is NOT interchangeable with `for` & `while` loops.
- Note that the initial value does not have to be assigned before the loop because in may be assigned in the body, which is done before the testing of the condition.

Common errors

Forget to make the change, or make a wrong change

- Forget to update the variable (i.e., no “change”)

```
int years = 1;
while(year<= 20) {
 double interest = balance * RATE / 100;
 balance = balance + interest;
}                                                year++;
```

- Increment instead of decrementing:

```
int years = 20;
while (years > 0) {
 double interest = balance * RATE / 100;
 balance = balance + interest;                   year--;
 years++;
}
```

Off-by-one errors

- Off-by-One Errors:
  - Should `years` start at 0 or 1?
  - Should the test be < or <= ?
  - Think, don't compile and try at random

```
int years = 0;
while (balance < targetBalance) {
   years++;
   balance = balance * (1 + RATE / 100);
}
```

- **Solution**: calculate the boundary cases, and trace by hand and/or use the debugger

REVIEW: `return`

Characteristics of `return`

- When a program encounters a `return` statement, it quits that method.
- If the method does not require an output (i.e., `void`), then (1) `return` simply quits the method, (2) `return` is NOT REQUIRED if the method ends at the closing brace of the method’s body.
- If the method requires an output/return, then (1) `return` must be followed by a value of the output/return data type, (2) `return` is REQUIRED, and is the last statement of the method for normal end.
- A statement that follows a `return` statement (in the same block) will cause a syntax error.
- A method that “returns” something means a method WITH OUTPUT; it does NOT mean a method that `print` or `println` something.

Example of `return` in a method w/o output

```
public void doSomething()  ← void indicate no output
{
    …
    if ( … ) {
        …
        return;  ← special-case return, in the middle of a method body
        // no more statements here, or syntax error
    }
    else
       …
    …
} ← normal return, at the ending brace of a method body
```

Example of `return` in a method w/ output

```
public double calcArea(double radius) ← double indicates with output, of type double
{
   double area = 0.0;
   if (radius < 0.0 )
      area = -1.0; // neg. area to indicate bad input
   else
     area = Math.PI * radius * radius;
   return area; ← REQUIRED return, and WITH output
   // no more statements here, or syntax error
}
```

REVIEW: `break`

Characteristics of `break`

- The `break` statement is used in a `switch` statement to end a `case`, at which point, the program goes to the statement after the `switch` body.
- The `break` statement is used in a loop to end the loop for a special case (i.e., not at the normal condition/test), at which point, the program continues at the statement after the loop’s body.
- A `break` statement, if used in a loop, must be inside a conditional (e.g., `if` or `switch`), or else the program will quit the loop at the first iteration.
- In a nested loop, a `break` statement will end the INNERMOST loop.

Terminating loops early

JC 6.5

break and return in Loops

- **break in a loop** instructs the program to immediately **quit the current iteration** and go to the **first statement following the loop**.
- **return in a loop** instructs the program to immediately **quit the current method** and **return to the calling method**.
- A break or return must be **inside an if or an else**, otherwise the code after it in the body of the loop will be unreachable.

break in Loops

- Example:

```
int d = n – 1;

while (d  > 0)
{
    if (n % d == 0)
        break; // quit the loop
    d––;
}

if ( d > 0 )   // if found a divisor
    ...
```

return in Loops

- Sequential Search method:

```
public int search(String[ ] list, String word)
{
    for (int k = 0;  k < list.length;  k++)
    {
        if (list[k].equals (word))
            return k;           // quit the method
    }

    return –1; // quit the method
}
```

to
calling
method

Some loop algorithms

JC 6.8

Min and max

- Find the min:

```
Initialize minVal ← first value
Do the following from the second value until the last value:
        curVal ← get the value
        if curVal < minVal , then there is a new min:
                curVal ← minVal
```

- Find the max – similar to the min, but the comparison is in opposite direction:

```
Initialize maxVal ← first value
Do the following from the second value until the last value:
        curVal ← get the value
        if curVal > maxVal , then there is a new max:
                curVal ← maxVal
```

- **NOTE**: initialize the min or max value to one of the data values; i.e., do NOT arbitrarily use a very large or very small number

Sum and average

- Calculate the total purchase at a check-out:

```
Initialize: total ← 0
Do the following until the last item is checked out:
          itemPrice ← scan/get the price for an item
          total ← total + itemPrice
```

- Calculate the average item price at a check-out – similar to the sum, plus counting the number of items in the loop, and divide outside the loop:

```
Initialize: total ← 0, count ← 0
Do the following until the last item is checked out:
          Scan/get the price for an item
          total ← total + itemPrice
          count ← count + 1
average ← total / count
```

Nested loops

JC 6.8

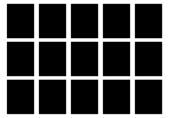
Nested Loops

- A loop within a loop is called nested.

```
// Draw a 5 by 3 grid:

for (int x = 0;  x < 50;  x += 10)
{
    for (int y = 0;  y < 30;  y += 10)
    {
        g.fillRect(x, y, 8, 8);
    }
}
```

Every 10 units, fill an 8x8 square



Increased by 10

Nested Loops (cont'd)

- Be careful with break:

```
for (int r = 0;  r < m.length;  r++)
{
    for (int c = 0;  c < m[0].length;  c++)
    {
        if (m [ r ][ c ] == 'X' )
            break;
    }
    // …. Some code not shown
}
...
```

Breaks out of the inner loop but <u>continues</u> with the outer loop

THE END