COMP-248
Object Oriented Programming I

Week 4: Control Flow 2

By Emad Shihab, PhD, Fall 2015,
Parts of the slides are taken from Prof. L. Kosseim
Adapted for Section EE by S. Ghaderpanah, Fall 2015

In this chapter, we will see:

1. The `if` statement
2. The `if-else` statement
3. Relations Operators
4. Logical operators
5. Compound statements
6. Nested `if` statements
7. The `switch` statement
8. The conditional operator
9. The `while` loop
10. The `do-while` loop
11. **The `for` loop**
12. Nested loops
13. break, continue & exit

11- The for loop

syntax:

Reserved word

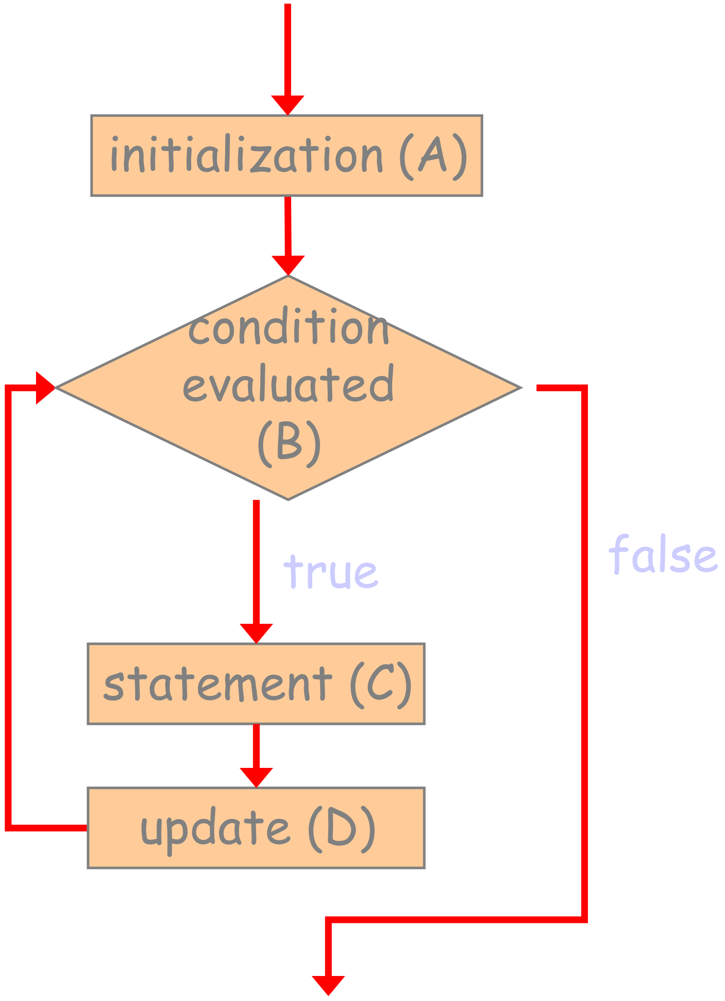
The *initialization* is executed once before the loop begins

The *statement* is executed until the *condition* becomes false

```
for ( initialization (A) ; condition (B); update (D))
    statement (C);
```

The *update* portion is executed at the end of each iteration
The *condition*-*statement*-*update* cycle is executed repeatedly

Logic of a for loop



Example

```
int i;
for (i=1; i<=5; i++)
    System.out.print(i);
System.out.print(i);
```

```
123456
```

Output

i
1
2
3
4
5
6

i increments to 6 the condition of the loop becomes false

Trace

for versus while

A `for` loop is equivalent to the following `while`:

```
initialization;
while ( condition )
{
    statement;
    update;
}
```

```
for ( initialization ; condition ; update )
    statement;
```

More examples

```
for (int i=0; i<0; i--)

    System.out.print("hello");
```

nothing

Output

```
for (int i=0; i<=0; i--)
    System.out.print("hello");
```

infinite loop

Output

Example: Display multiples

```
Enter a positive value:  10
Enter an upper limit:  95

Multiples of 10 between 10 and 95:
10 20 30 40 50
60 70 80 90
```

Output

Data needed:

Algorithm:

Example: Display multiples

Multiples.java

```
final int PER_LINE = 5;
int value, limit, mult, count = 0;
Scanner myKeyboard = new Scanner(System.in);
System.out.print("Enter a positive value: ");
value = myKeyboard.nextInt();
System.out.print("Enter an upper limit: ");
limit = myKeyboard.nextInt();

System.out.println("Multiples of "+value+" between "+ value + " & " + limit);

for ( mult=value ; mult<=limit ; mult+=10 ) {
   System.out.print(mult + "\t");

   // Print a specific number of values per line of output
   count++;
   if (count % PER_LINE == 0)
      System.out.println();
}
```

Example: Display multiples

```
Enter a positive value:  10
Enter an upper limit:  95

Multiples of 10 between 10 and 95:
10 20 30 40 50
60 70 80 90
```

Output

Data needed:

Algorithm:

Improved solution working with any positive start value

```java
public static void multiples(int startValue, int upperLimit) {
    for(int i=0; i<=upperLimit; i+=10) {
        if(i>=startValue)
            System.out.println(i);
    }
}
```

More on for loops

Each expression in the header of a `for` loop is optional

- If the ***`initialization`*** is left out
  - no initialization is performed
- If the ***`condition`*** is left out
  - it is always considered to be true
- If the ***`update`*** is left out
  - no update operation is performed

Both semi-colons are always required

Just checking ...

Which of the loops below produces the same number of loop iterations as the following loop? (**`count`** is of type **`int`**)

```
for (count = 1; count <= 10; count++)
     whatever…
```

```
A.   for (count = 10; count >= 1; count--)
        whatever

B.   for (count = 0; count < 10; count++)
        whatever

C.   for (count = 10; count >= 0; count--)
        whatever
```

D. A and B above correct

E. A, B and C above false, C produces 11 iterations

Even more on for loops

- *initialization*
- *update*

```
for (int i=0, j = 1; i<=10; i++, j=2*j)
    System.out.print(i + " " + j);
```

Example

Assume:

```
    int sum, i;
```

```
sum = 0;
for (i=0; i<5; i++)
    sum+=i;
System.out.print(sum);
```

10 = 0+1+2+3+4

Output

```
for (i=0, sum=0; i<5; sum+=i,i++)
    ;
System.out.print(sum);
```

10 again

Output

```
for (i=0,sum=0; i<5; i++)
    sum+=i;
System.out.print(sum);
```

10 again

Output

```
for (i=0,sum=0; i<5; i++,sum+=i)
    ;
System.out.print(sum);
```

15 = 1 + 2 + 3 + 4 + 5

Output

i is incremented before sum

Which loop to use?

any loop can be re-written with another loop

in general, use a:

- **`while`** or a **`do-while`**:
  - when you don't know in advance how many times you want to execute the loop body
  - if it will be at least once, use a **do** loop
- **`for`**:
  - when you know how many times you want to execute the loop body

In this chapter, we will see:

1. The `if` statement
2. The `if-else` statement
3. Relations Operators
4. Logical operators
5. Compound statements
6. Nested `if` statements
7. The `switch` statement
8. The conditional operator
9. The `while` loop
10. The `do-while` loop
11. The `for` loop
12. **Nested loops**
13. break, continue & exit

12- Nested loops

a **`for`** inside a **`for`**, a **`while`** inside a **`for`**, a **`do-while`** inside a **`while`**, …

i.e. the body of a loop can contain another loop

consists of:

- an outer loop
- an inner loop

for 1 iteration of the outer loop, the inner loop goes through its full set of iterations

Example

```
for (i = 2; i <= 4; i++)
{
   for (j = 6; j <= 7; j++)
      System.out.println(i + " " + j);

   System.out.println("j is now " + j);
}
System.out.println("i is now " + i);
```

```
2 6
2 7
j is now 8
3 6
3 7
j is now 8
4 6
4 7
j is now 8
i is now 5
```

Output

i	j
2	6
2	7
2	8
3	6
3	7
3	8
4	6
4	7
4	8
5	8

Trace

Another example

```
int numRows, r, i;
numRows = keyboard.nextInt();
for (r = 1; r <= numRows; r++)
 {
    for (i = 1; i <= numRows-r; i++)
       System.out.print(" ");

    for (i = 1; i <= r; i++)
       System.out.print("*");

    System.out.println();
 }
```

```
  *
 **
***
```

Output

numRows	r	i
3	1	1
3	1	2
3	1	1
3	2	1
3	2	1
3	2	2
3	3	1
3	3	2
3	3	3

Trace

Example

```
1   2   3   4   5
2   3   4   5
3   4   5
4   5

5
```

Output

algorithm??

Java code ??

```
public static void printNumbers() {
    for(int i=1; i<=5; i++) {
        for(int j=i; j<=5; j++) {
            System.out.print(j + "\t");
        }
        System.out.println();
    }
}
```

What is the output?

```
int n = 2;
for (int loopCount = 1; loopCount <= 3; loopCount++)
   while (n <= 4)
      n = 2 * n;
System.out.println(n);
```

A. 4

B. 8

C. 16

D. 32

E. 64

Correct, the while loop is executed only 2 times in the 1st iteration of the for loop

In this chapter, we will see:

1. The `if` statement
2. The `if-else` statement
3. Relations Operators
4. Logical operators
5. Compound statements
6. Nested `if` statements
7. The `switch` statement
8. The conditional operator
9. The `while` loop
10. The `do-while` loop
11. The `for` loop
12. Nested loops
13. **break, continue & exit**

13- break and continue

bypasses the normal flow of control of loops

very practical sometimes… but use in moderation…

`break`

- will exit the inner-most loop without evaluating the condition

`continue`

- will interrupt the current iteration (of the inner-most loop)
- and will force a new evaluation of the condition for a possible new iteration
- Note: in a for loop, the incrementation is done before the condition is tested…

Example

```
int n;
while (true) {
   System.out.print("Enter a positive integer");
   n = keyboard.nextInt();
   if (n < 0)
     break;
   System.out.println("squareroot of " + n + " = " + Math.sqrt(n));
 }
```

```
int n;
while (true) {
   System.out.print("Enter a positive integer or 0 to exit");
   n = keyboard.nextInt();
   if (n == 0)
     break;
   if (n < 0)
     continue;
   System.out.println("squareroot of " + n + " = " + Math.sqrt(n));
 }
```

Ex. Prime numbers from 10 to 50

Prime.java

```
11 13 17 19 23 …
```

Output

```
10 --> verify 2 3 4 5 6 7 8 9
11 --> verify 2 3 4 5 6 7 8 9 10
12 --> verify 2 3 4 … 11
…
15 --> verify 2 3 4 5 … 14
…
33 --> verify 2 3 4 5 … 32
…
50 --> verify 2 3 4 … 49
```

Method

```
boolean divisible;
final int UP = 50;
final int LOW = 10;

for (int number = LOW; number <= UP; number++)
{
   for (int candidate = 2; candidate < number; candidate++)
   {
      divisible = (number % candidate) == 0;
      if (divisible)
         break; ??  continue; ??
   }
   if (!divisible) // ok ?
      System.out.print(number + " ");
}
```

Correct is break;

Code

The exit Statement

- A `break` statement will end a loop or switch statement, but will not end the program
- The `exit` statement will immediately end the program as soon as it is invoked:

```
System.exit(0);
```

- The `exit` statement takes one integer argument
  - By tradition, a zero argument is used to indicate a normal ending of the program

In this chapter, we have seen:

1. The `if` statement
2. The `if-else` statement
3. Relations Operators
4. Logical operators
5. Compound statements
6. Nested `if` statements
7. The `switch` statement
8. The conditional operator
9. The `while` loop
10. The `do-while` loop
11. The `for` loop
12. Nested loops
13. break, continue & exit