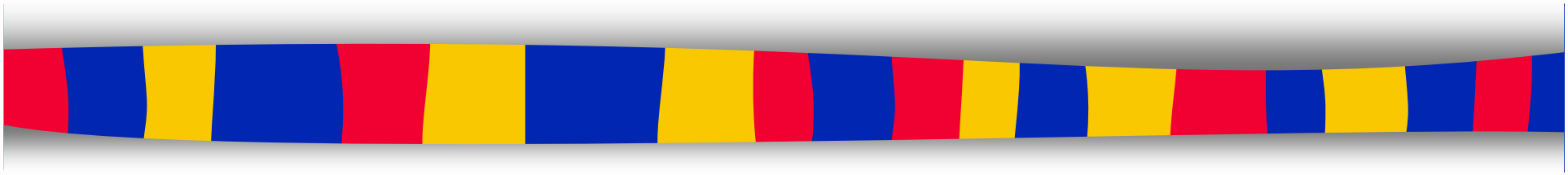
COMP-248
Object Oriented Programming I



Week 3: 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

4- Logical operators

To combine multiple boolean expressions into a more complex one

`!`	Logical NOT	(unary operator)
`&&`	Logical AND	(binary operator))
`||`	Logical OR	(binary operator)

They all take boolean operands and produce boolean results

`!a` is **`false`** if `a` is **`true`**
`!a` is **`true`** if `a` is **`false`**

a	!a
true	
false	

4- Logical operators

a && b is **true** only if **both a** and **b** are **true**
a && b is **false** if **a** or **b** or both are **false**

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

4- Logical operators

a || b is **true** if **a** or **b** or both are **true**
a || b is **false** if **both a** and **b** are **false**

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

Just checking …

When using a compound Boolean expression joined by an && (AND) in an if statement:

A. Both expressions must evaluate to true for the statement to execute.

B. The first expression must evaluate to true and the second expression must evaluate to false for the statement to execute.

C. The first expression must evaluate to false and the second expression must evaluate to true for the statement to execute.

D. Both expressions must evaluate to false for the statement to execute.

Just checking …

If **p** is a Boolean variable, which of the following logical expressions always has the value **false**?

A. `p && p`
B. `p || p`
C. `p && !p`
D. `p || !p`
E. b and d above

Precedence and Associativity Rules

Boolean and arithmetic expressions need not be fully parenthesized

If parentheses are omitted, Java follows ***precedence*** **and** ***associativity*** rules to determine the order of operations

- If one operator has *higher precedence*
  - it is grouped with its operands before the operator of lower precedence
- If two operators have the same precedence
  - then *associativity rules* determine which is grouped first

Precedence and Associativity Rules

Precedence	Operator	Associativity
highest (evaluated 1st)	postfix ++, postfix --	right to left
	unary +, unary -, prefix ++, prefix --, !	right to left
	type casts	
	binary *, / %	left to right
	binary +, -	left to right
	binary >, <, >=, <=	left to right
	binary ==, !=	left to right
	binary &	left to right
	binary |	left to right
	binary &&	left to right
	binary ||	left to right
	conditional operator ?:	right to left
lowest (evaluated last)	assignment operators: =, *= /=, %=, +=, -=, &=, |=	right to left

Logical operators

```
if (total < MAX+5 && !found)
   System.out.println ("Processing…");
```

```
if ((total < (MAX+5)) && (!found))
   System.out.println ("Processing…");
```

precedence:

- all logical operators have lower precedence than the relational or arithmetic operators
- logical NOT has higher precedence than logical AND and logical OR

Short-circuit evaluation

```
if (count!=0 && total/count > MAX)
   System.out.println ("Testing…");
```

if ***(count!=0)*** *is false… no point in evaluating* ***(total/count>MAX)***

```
if (isChild || height < 1.5)
 System.out.println ("half price");
```

- *if (isChild) is true… no point in evaluating (height*

The processing of logical AND and logical OR is "short-circuited"

also called *lazy evaluation*

If the left operand is sufficient to determine the result of the entire condition, the right operand is **not** evaluated

Just checking

Does the following sequence produce a division by zero?

```
int j = -1;
if ((j > 0) && (1/(j+1) > 10));
   System.out.println(i);
```

A. Yes, this sequence produces a division by zero.

B. No, this sequence does not produce division by zero.

C. We have no way of knowing.

D. No, this sequence does not produce division by zero, because it has a syntax error.

Complete evaluation

to force both expressions to be evaluated

- use & instead of &&
- use | instead of ||

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

5- Compound statements

```
if ( condition )
   statement;
```

```
if ( condition )
   statement1;
else
   statement2;
```

what if you wanted to execute several statements?

Several statements can be grouped together into a ***compound statement*** (or block):

```
{
   statement1;
   statement2;
   …
}
```

A **block** can be used wherever a statement is called for by the Java syntax

Example

```
int grade;

System.out.print("what is your grade?");
grade = myKeyboard.nextInt();

if (grade >= 80)
   System.out.println("congratulations!");
else
   {
   System.out.println("you could do better");
   System.out.println("make sure you practice");
   }
System.out.println("bye bye");
```

89?

79?

Output

Next topic:

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