Conditional Statements

Programming in Java

Contents

2- The **if-else** statement: Exercises

- Write a Java program that reads a floating-point number and prints "zero" if the number is zero. Otherwise, print "positive" or "negative". Add "small" if the absolute value of the number is less than 1, or "large" if it exceeds 1,000,000.

- Write a Java program that reads in two floating-point numbers and tests whether they are the same up to three decimal places.
- Write a Java program to find the number of days in a month.

3- Relational Operators

- Needed in control structures (ex. if)
- To write conditions (boolean expressions)
- Return boolean results (evaluates to true or false)

==	equal to
!=	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

- Note the difference between == and =

3- Relational Operators: Example (IncomeTax.java)

```
if (age == 18)
    System.out.println("you are 18");
else
    System.out.println("you are not 18");
```

Output

```
if (age = 18)
    System.out.println("you are 18");
else
    System.out.println("you are not 18");
```

Output

3- Relational Operators: A note on comparing floats

- Be careful when comparing 2 floating point values (**float** or **double**) for equality
- Do not use the equality operator (==)
- Because floats are approximated
- You want to see if two floats are "close enough"

```
if (Math.abs(f1 - f2) < 0.00001)
   System.out.println ("Essentially equal.");
```

3- Relational Operators: A note on comparing characters

- We can use the relational operators to compare 2 characters
- The results are based on the Unicode character set

```
if ('+' < 'J')
   System.out.println("+ is less than J in Unicode");
```

```
char userAnswer = 'n';
if (userAnswer == 'N')
   System.out.println("the user said no");
```

3- Relational Operators: A note on comparing strings

- We **cannot** use the relational operators to compare strings (<, ==, …)

- Use the **equals()** method
  - to determine if two strings have the same content
  - ex: **firstString.equals(secondString)**
  - returns a **boolean**:
    - **true** if **firstString** has the same content as **secondString**
    - **false** otherwise

3- Relational Operators: A note on comparing strings

- Use the **compareTo()** method
  - to determine if one string comes before another (based on the Unicode character set)
  - ex: **firstString.compareTo(secondString)** returns an **int**:
    - **negative** if **firstString** is lexicographically before **secondString**
    - **positive** if **firstString** is lexicographically after **secondString**
    - **0** if the two strings have the same content

3- Relational Operators: Example 1 (StringComparison.java)

```
String s1 = "Java isn't just for breakfast.";
String s2 = "JAVA isn't just for breakfast.";

  if (s1.equals(s2))
    System.out.println("The two lines are equal.");
  else
    System.out.println("The two lines are not equal.");


  if (s2.equals(s1))
    System.out.println("The two lines are equal.");
  else
    System.out.println("The two lines are not equal.");
```

3- Relational Operators: Example 1 (StringComparison.java)

```
String s1 = "Java isn't just for breakfast.";
String s2 = "JAVA isn't just for breakfast.";

if (s1.equalsIgnoreCase(s2))
 System.out.println("But the lines are equal, "
       + " ignoring case.");
else
 System.out.println("Lines are not equal, "
       + " even ignoring case.");
```

3- Relational Operators: Example 2

	Syntax Error?	Output?
`System.out.println("aBcD" < "abcd");`		
`System.out.println('aBcD' < 'abcd');`		
`System.out.println("a" < "b");`		
`System.out.println('a' < 'b');`		
`System.out.println("aBcD".equals("abcd"));`		

3- Relational Operators: Example 2

	Syntax Error?	Output?
`System.out.println("aBcD".equalsIgnoreCase("aBcD"));`		
`System.out.println("aBcD".compareTo("aBcD"));`		
`System.out.println("aBcD".compareTo("aBcC"));`		
`System.out.println("abc".compareTo("ab"));`		
`System.out.println("abc".compareTo("abcd"));`		

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	false	
false	true	
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	false	
false	true	
false	false	

4- Logical Operators: Exercise

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.

4- Logical Operators: Exercise

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

4- Logical Operators: 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

4- Logical Operators: 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

4- Logical Operators: Exercise

```
int a = 10;

int b = ++a++;
```

Which is the correct option?

A. a = 12 and b = 12

B. b = 12 and b = 11

C. Compiler error

D. Syntax error

4- 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

4- Logical Operators: 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 < 1.5)
```

- 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.

4- Logical Operators: Exercise

Does the following sequence produce a division by zero?

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

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

4- Logical Operators: Complete evaluation

- To force **both** expressions to be evaluated
  - use & instead of &&
  - use | instead of ||
- Rarely used …

5- Compound statements

- So far have seen

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

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

- What if you wanted to execute several statements?

5- Compound 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

5- Compound statements: 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");
```

6- Nested if statements

- An **if** is a statement… so we can put an **if** “inside” an **if**

- Called nested **if statements**

```
if ( condition1 )
      if(condition2 )
        statement;
      else
       statement;
```

6- Nested if statements

```
if ( condition1 )
{
      if(condition2 )
        statement1;

      statement2;
}
```

6- Nested if statements

```
if ( condition1 )
      if(condition2 )
        {
         statement1;
         statement2;
        }
      else
       statement;
```

6- Nested if statements: Example

```
int num1, num2, num3, min = 0;
System.out.println ("Enter three integers: ");
num1 = keyboard.nextInt();
num2 = keyboard.nextInt();
num3 = keyboard.nextInt();
if (num1 < num2)
    if (num1 < num3)
       min = num1;
    else
       min = num3;
else
    if (num2 < num3)
       min = num2;
    else
       min = num3;
System.out.println ("Minimum value: " + min);
```

6- Nested if statements

- Given the following code segment, what is stored in a at the end of this sequence?

```
int a = 0;
if (a >= 10)
if (a < 20)
a = a + 2;
else
a = a + 1;
```

A. 0
B. 1
C. 2
D. 3
E. Syntax error

See why indenting makes it easier to read code!

6- Nested if statements

- An **else clause** is matched to the last unmatched if (no matter what the indentation)

```
if (condition1)
    if (condition2)
        statement1;
else
        statement2;
```

```
if (condition1)
{
  if (condition2)
    statement1;
}
else
  statement2;
```

6- Nested if statements: Exercise: Leap year

- **Problem:**
  - Leap years occur in years exactly divisible by four, except that years ending in 00 are leap years only if they are divisible by 400.
- **Example:**
  - 1700, 1800, 1900, 2100, and 2200 are not leap years
  - 1600, 2000, and 2400 are leap years.
- **Algorithm:**
  - if year is a multiple of 400 ----> leap
  - otherwise
    - if year is a multiple of 100 ----> not leap
    - otherwise
      - if year is a multiple of 4 ----> leap
      - otherwise ----> not leap