More Conditional Statements

More about if, switch, and boolean operations.

Simple Boolean Expressions

A **boolean expression** is anything with a value true or false.

- Comparisons of primitive data types:

```
x > y        x >= y
x < y        x <= y
x == y       x != y
```

- Comparison of *object values*:

```
 obj1.equals( obj2 )
! obj1.equals( obj2 )
```

- Comparison of *object references*:

```
obj1 == obj2
```

is true of **`obj1`** and **`obj2`** *refer to the same object.*

Comparison of Object References

- Example of common error using == for objects:

```
Double x = new Double(10.0);
Double y = new Double(10.0);
if ( x == y ) System.out.println("x == y");
else System.out.println("x != y");
```

```
output: x != y
```

- Use **equals** to compare the *value of objects*:

```
Double x = new Double(10.0);
Double y = new Double(10.0);
if ( x.equals(y) ) System.out.println("x equal y");
else System.out.println("x not equal y");
```

```
output: x equal y
```

Compound Boolean Expressions

- Compound Operations:

`A && B` true if A is true *and* B is true

`A || B` true if A is true *or* B is true

`A ^ B` true if A or B is true, but not both true

```
> true ^ false
true
> true ^ true
false
> false ^ false
false
```

Short-Circuit Evaluation

Stop as soon as the result is known:

`A && B` if A is **false**, don't test B

`A || B` if A is **true**, don't test B

Example:

```
// avoid division by zero
if ( y != 0 && (x/y) < 1 ) ... ;
```

(x/y) is not performed if y == 0.

Short-Circuit and Function Calls

Short-circuit evaluation can be useful to avoid unnecessary function calls.

Example:

```
// if string is null or length zero
// then print "invalid"
if (string == null
        || string.length()==0) ...
```

This is not performed if string==null

Boolean Logic

- Negation: `!` ***`expression`***

  `!( x == y )` is same as **`(x != y)`**

  `!(A && B)` is same as **`!A || !B`**

  `!(A || B)` is same as **`!A && !B`**

- Applies to any number of conjunctions:

  `!(A && B && C)` is **`!A || !B || !C`**

  `!(A || B || C)` is **`!A && !B && !C`**

`What about: !(A && B || C)`

&, |, ^ always evaluate both args

- **&, |, ^** are **bitwise operators** that apply to boolean and other types.
- they always evaluate both expressions:

A & B *A and B*, always evaluates A, B

A | B *A or B*, always evaluates A, B

A ^ B *exclusive or*, always evaluates A, B

Truth Table

A	B	A & B	A | B	A ^ B	! (A & B)
true	true	true	true	false	false
true	false	false	true	true	true
false	true	false	true	true	true
false	false	false	false	false	true

Examples

Copy Center

- The copy center (KU's most popular service) charges according to the number of copies:

1 - 9 copies	0.50 baht/copy
10-49 copies	0.45 baht/copy
50-99 copies	0.42 baht/copy
100+ copies	0.40 baht/copy

- Complete this method for computing copy charges:

```
/** compute price of copy job.
 *  @param copies = number of copies in job
 */
public static double jobCost(int copies) {




}
```

Copy Center (2)

- The rate table:

1 - 9 copies	0.50 baht/copy
10-49 copies	0.45 baht/copy
50-99 copies	0.42 baht/copy
100+ copies	0.40 baht/copy

- Simple solution:

```
/** compute price of copy job.
 *  @param copies = number of copies in job
 */
public static double jobCost(int copies) {
   double price;
   if ( copies < 10 ) price = 0.50*copies;
   if ( copies >= 10 && copies < 50 ) price = 0.45*copies;
   if ( copies >= 50 && copies <100 ) price = 0.42*copies;
   if ( copies >= 100 ) price = 0.40*copies;
   return price;
}
```

Copy Center (3)

- This is inefficient because it performs redundant tests.
- Better answer:

```
/** compute price of copy job.
 *  @param copies = number of copies in job
 */
public static double jobCost(int copies) {
   double price;
   if ( copies < 10 ) price = 0.50*copies;
   else if ( copies < 50 ) price = 0.45*copies;
   else if (  copies < 100 ) price = 0.42*copies;
   else price = 0.40*copies;
   return price;
}
```

- But there is a logic error here (missing case). What?

Copy Center (4)

- Be careful of the case copies < 0. Two solutions:
  - Require the caller verify the data:

    ```
    @precondition  copies >= 0
    ```

  - Check for the case copies < 0.

```
/** compute price of copy job.
 *  @param copies = number of copies in job
 */
public static double jobCost(int copies) {
   double price;
   if ( copies < 0 ) return 0.0;
   ...
}
```

Testing Yes/No Input

```
String reply = input.next( ); // read a word
if ( reply == null ) /* do nothing */ ;
else if ( reply.equalsIgnoreCase("yes") ) ... ;
else if ( reply.equalsIgnoreCase("no") ) ... ;
else System.out.println("what?");
```

This works because Scanner.next( ) trims leading and trailing blanks from the input word.

If you use some other input method, you should use reply.trim( ) to trim leading and trailing blanks before testing the reply.

Common Errors

- This example contains a syntax error and a logic error.

```
// binomial formula for a*x^2 + b*x + c = 0
double discrim = b*b - 4*a*c;
if ( discrim > 0 );
{     r = Math.sqrt( discrim );
      x = ...;  /* compute a root */
}
else System.out.println("No real roots");
```

Bitwise operators

Bitwise operators compare or manipulate bits on primitive data types.

Bitwise Logic Operators

&	AND
|	OR
^	XOR
~	bitwise NOT (ones complement)

Shift operators

<<	left-shift
>>	right-shift
>>>	unsigned right shift

Bitwise operators

& (Bitwise AND)

	0	1
0	0	0
1	0	1

| (Bitwise OR)

	0	1
0	0	1
1	1	1

^ (Bitwise XOR)

	0	1
0	0	1
1	1	0

~ (Bitwise NOT)

0	1
1	0

Bitwise operators

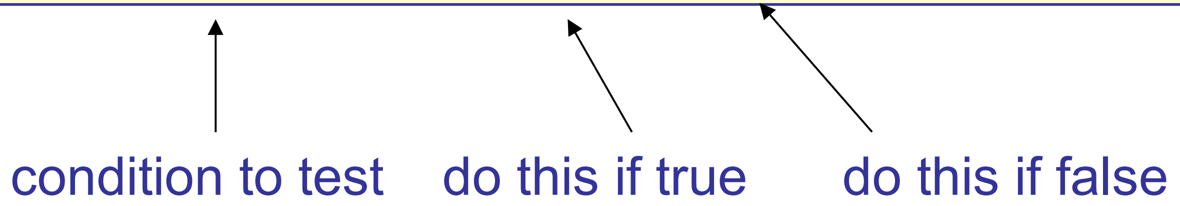
Crude way to view the bit representation of an integer:

```
int n = 12345;
// show the bit in the 4th position from right
bit = n & (1<<3);  //  = n & 000001000

// shift this bit all the way to the right
bit = (n & (1<<3)) >> 3; // = 1 if 4th bit of n is 1

// show all the bits using a loop
for( int k = Integer.SIZE -1; k>=0; k-- ) {
      bit = (n & (1 << k)) >> k;
      System.out.print("" + bit);
}
System.out.println("");
```

(condition) ? expression1 : expression2

An inline version of “if … else ...”.
The only ternary (3 argument) operator in Java. The usage is:

```
String grade;
grade = ( score > 60 ) ? “pass” : “fail”;
```



condition to test do this if true do this if false

```
// is the same as this…
if ( score > 60 ) grade = “pass”;
else grade = “fail” ;
```

Conditional Examples

```
// Announce new mail:
int numMessages = getNewMail( );
System.out.println("You have " + numMessages
          + " new " +
  (numMessages == 1 ? "message" : "messages") );
```

```
You have 1 new message      if numMessages == 1
You have 3 new messages     any other value
```

switch for multiple alternatives

```
reply = (char) System.in.read( );  // reply to y(es) or n(o) question
switch ( reply ) {
   case ‘y’:
   case ‘Y’:
       println(“that was yes”);
       break;
   case ‘n’:
   case ‘N’:
       println(“that was no”);
       break;
   default:
       println(“invalid reply”);
}
```

Two cases execute the same code.

Two cases execute the same code.

#1 Coding Error: forgetting "break"

"switch" is same as compound “if”

The previous “select” example is the same as:

```
if ( reply == ‘y’ || reply == ‘Y’ ) {
          System.out.println(“that was yes”);
}
else if ( reply == ‘n’ || reply == ‘N’ ) {
          System.out.println(“that was no”);
}
else {:
          System.out.println(“invalid reply”);
}
```

Syntax of the switch Statement

```
switch ( expression ) { // Start switch block
 case value1:
         statement;
case value2:
          statement;
          statement;
 case value3:
          ...
 default:
          statement;
} // end of switch block
```

compare *expression* to each of the case values: go to the first one that matches.
Then **continue** until the end of switch statement
or a "**break**" is reached.

If no matches, then do the "default" case (optional).

The ***expression*** can be of type **char**, **byte**, **short**, or **int**.
Starting in Java 7, String is allowed, too.

Using "break" in switch

```
switch ( expression ) { // Start switch block
     case value1:
          statement;
     case value2:
          statement;
          statement;
          break;
     case value3:
          . . .
} // end of switch block
next_statement;
```

when break is encountered, control will jump to the instruction *after* end of the switch block.

break causes execution to "break out" of a switch or loop.

break causes execution to jump forward to the end of block.

Switch Example

```
switch ( grade ) { // Start switch block
case 'A':
    gp = 4.0;
    break;
case 'B':
    gp = 3.0;
    break;
case 'C':
    gp = 2.0;
    break;
default:
    gp = 1.0;
} // end of switch block
System.out.println("gp = "+gp);
```

Switch with Strings

Starting in Java 7, a switch expression may be a String.

```
System.out.print("Do you like Java? ");
String answer = scanner.next().toLowerCase();
switch ( answer ) { // switch using string
case "yes":
    System.out.println("Great! Try Kotlin, too.");
    break;
case "no":
    System.out.println("Try Python instead.");
    break;
default:
    System.out.println("Try a typing course");
}
```