CMSC 131: Chapter 29 (Supplement)
Miscellany II

Operators Revisited

Operators: We discussed various operators (+, -, *, <, ==, &&, ||) earlier this semester. We
omitted discussion of a few.

Bitwise operators: Operate on values as binary numbers.
Conditional operator: An "if-then-else" operator.

Bitwise Operators: Recall that all quantities are stored as binary numbers in memory. For
example:

```java
int x = 1037;  // binary: ...0010000011101 filled out to 32 bits
char c = 'Y';  // binary: ...00000011110011 filled out to 16 bits
boolean b = true;  // binary: 1
```

You are not required to know how these conversions are performed. (It is covered in
later courses.)

Java's bitwise operators act on these binary representations.

Bitwise Operators

Java supports the standard bit operators:

~a: complement of a
a & b: and (1 if both a and b are 1)
a | b: or (1 if either a and b are 1)
a ^ b: exclusive or (1 if either a or b is 1, but not both)
Bitwise Operators

Java's bitwise operators can be applied
- to any integral type: char, byte, short, int, long
- to Boolean

When applied to integral types, the operations are applied bitwise:

```java
int a = 45; // a = 00101101
int b = 14; // b = 00001110
int c = a & b; // c = (00101101 & 00001110) = 00001110 (= 12)
```

Who uses these: They are used in often hardware-related tasks (device management) and have other surprising uses. (E.g.: Using exclusive-or you can swap to integers without a temporary.)

Shift Operators

Another common operation involves **shifting** bits left or right.

- a << b: Shift a left by b positions
- a >> b: Shift a right by b positions (filling with the sign bit)
- a >>> b: Shift a right by b positions (filling with 0s)

Notes:
- a must be integral type (byte, short, ..., long).
- b should be a nonnegative integral type.

Sign bit: Because there is no "-" sign in binary, Java encodes negative numbers using a method called 2's-complement representation. We will not discuss this, but a key element is that the leftmost bit, called the **sign bit**, is:

- 0 for positive numbers
- 1 for negative numbers

We often want to keep the sign bit **unchanged** when shifting.
Shift Operators

Example: Rather than use 32-bit int’s, we use a 10-bit example.

```java
int a = ...  // a = 1100101101
int b = 3;
int c = a << b;  // c = 0101101000
int d = a >> b;  // d = 1111100101
int e = a >>> b;  // e = 0001100101
int f = ...  // f = 0100101101
int g = f >> b;  // g = 0000100101
```

Conditional Operator

Conditional Operator: This is a handy operator, which acts like a little if-then-else statement within an expression.

```java
(boolean-condition) ? (true-value) : (false-value)
```

Example: Absolute value method. If the argument x is negative, then return -x, otherwise return x. Without conditional:

```java
public static int absValue1( int x ){
   if ( x < 0 ) return -x;
   else return x;
}
```

With the conditional operator: you can put the whole expression on one line.

```java
public static int absValue2( int x ){
   return ( x < 0 ? -x : x );
}
```

Conditional Operator

Example: Set max to the maximum of x and y.

```java
double max = (x > y) ? x : y ;
```

Example: If String s is "zero", then set x to 0.0, and otherwise set it to 3*(z + 13.2):

```java
String s = ...
double z = ...
double x = s.equals("zero") ? 0.0 : 3*(z + 13.2);
```