



Advanced if/else & Cumulative Sum

Subset of the Supplement Lesson slides from: Building Java Programs, Chapter 4
by Stuart Reges and Marty Stepp (<http://www.buildingjavaprograms.com/>) & thanks to Ms Martin.

Questions to consider

- What are the advantages of using Returns?
- What do we have to consider when returning a value in a series of nested if/else's?
- What additional Operators do we need to make our if conditions (tests) more useful?

if/else with return

```
// Returns the larger of the two given integers.
public static int max(int a, int b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
```

- Methods can return different values using `if/else`
 - Whichever path the code enters, it will return that value.
 - Returning a value causes a method to immediately exit.
 - All paths through the code must reach a `return` statement.

All paths must return

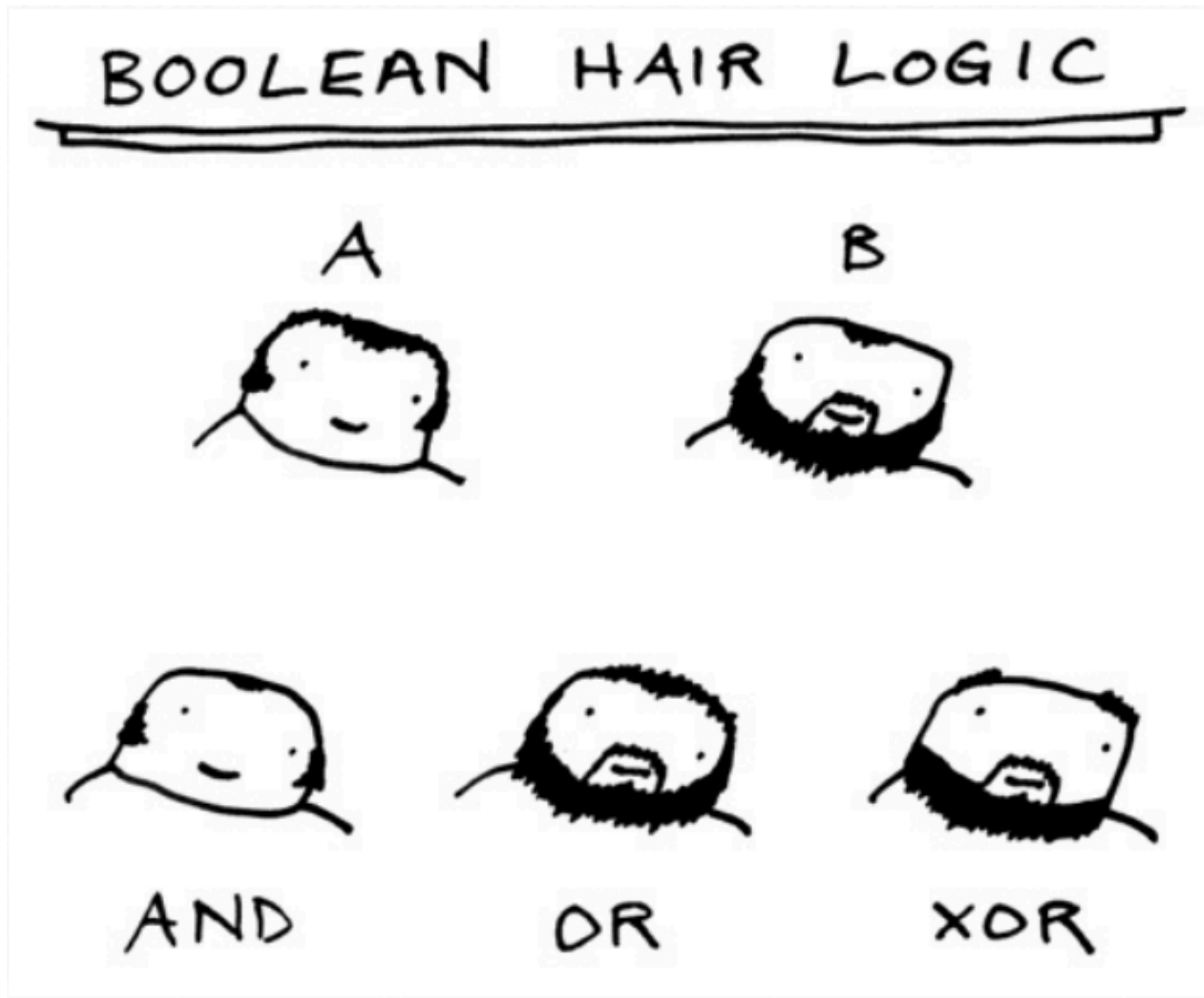
```
public static int max(int a, int b) {  
    if (a > b) {  
        return a;  
    }  
    // Error: not all paths return a value  
}
```

- The following also does not compile:

```
public static int max(int a, int b) {  
    if (a > b) {  
        return a;  
    } else if (b >= a) {  
        return b;  
    }  
}
```

- The compiler thinks `if/else/if` code might skip all paths, even though mathematically it must choose one or the other.

Logic



Logical operators

- Tests can be combined using *logical operators*:

Operator	Description	Example	Result
&&	and	(2 == 3) && (-1 < 5)	false
	or	(2 == 3) (-1 < 5)	true
!	not	!(2 == 3)	true

- "Truth tables" for each, used with logical values p and q :

p	q	p && q	p q
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

p	!p
true	false
false	true

Evaluating logic expressions

- Relational operators have lower precedence than math.

```
5 * 7 >= 3 + 5 * (7 - 1)
5 * 7 >= 3 + 5 * 6
35      >= 3 + 30
35      >= 33
true
```

- Relational operators cannot be "chained" as in algebra.

```
2 <= x <= 10
true      <= 10      (assume that x is 15)
error!
```

- Instead, combine multiple tests with `&&` or `||`

```
2 <= x && x <= 10
true && false
false
```

Logical questions

- What is the result of each of the following expressions?

```
int x = 42;  
int y = 17;  
int z = 25;
```

- `y < x && y <= z`
- `x % 2 == y % 2 || x % 2 == z % 2`
- `x <= y + z && x >= y + z`
- `!(x < y && x < z)`
- `(x + y) % 2 == 0 || !((z - y) % 2 == 0)`

- **Answers:** `true, false, true, true, false`

- **Exercise:** Write a program that prompts for information about a person and uses it to decide whether to date them.

Factoring if/else code

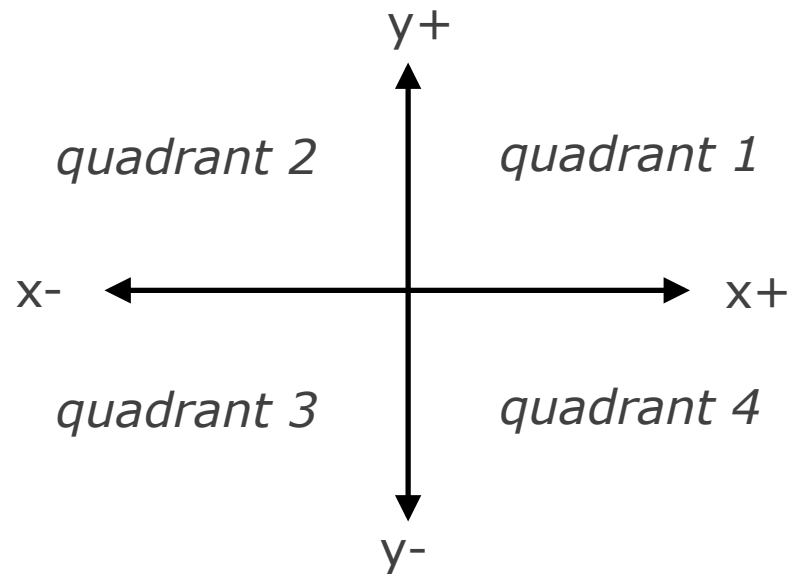
- **factoring:** Extracting common/redundant code.
 - Can reduce or eliminate redundancy from `if/else` code.
- **Example:**

```
if (a == 1) {  
    System.out.println(a);  
    x = 3;  
    b = b + x;  
} else if (a == 2) {  
    System.out.println(a);  
    x = 6;  
    y = y + 10;  
    b = b + x;  
} else { // a == 3  
    System.out.println(a);  
    x = 9;  
    b = b + x;  
}
```

```
System.out.println(a);  
x = 3 * a;  
if (a == 2) {  
    y = y + 10;  
}  
b = b + x;
```

if/else, return question

- Write a method `quadrant` that accepts a pair of real numbers x and y and returns the quadrant for that point:



- Example: `quadrant(-4.2, 17.3)` returns 2
 - If the point falls directly on either axis, return 0.

if/else, return answer

```
public static int quadrant(double x, double y) {
    if (x > 0 && y > 0) {
        return 1;
    } else if (x < 0 && y > 0) {
        return 2;
    } else if (x < 0 && y < 0) {
        return 3;
    } else if (x > 0 && y < 0) {
        return 4;
    } else {          // at least one coordinate equals 0
        return 0;
    }
}
```

Code Sample Example

- Write a method `daysInMonth` that accepts an integer representing the month and returns the number of days in that month.
- Assume there are no leap years

Month	1 Jan	2 Feb	3 Mar	4 Apr	5 May	6 Jun	7 Jul	8 Aug	9 Sep	10 Oct	11 Nov	12 Dec
Days	31	28	31	30	31	30	31	31	30	31	30	31

- Examples:

`daysInMonth(2)` returns 28

`daysInMonth(5)` returns 31



Cumulative algorithms

Adding many numbers

- How would you find the sum of all integers from 1-1000?

```
// This may require a lot of typing  
int sum = 1 + 2 + 3 + 4 + ... ;  
System.out.println("The sum is " + sum);
```

- What if we want the sum from 1 - 1,000,000?
Or the sum up to any maximum?
 - How can we generalize the above code?

Cumulative sum loop

```
int sum = 0;
for (int i = 1; i <= 1000; i++) {
    sum = sum + i;
}
System.out.println("The sum is " + sum);
```

- **cumulative sum:** A variable that keeps a sum in progress and is updated repeatedly until summing is finished.
 - The `sum` in the above code is an attempt at a cumulative sum.
 - Cumulative sum variables must be declared *outside* the loops that update them, so that they will still exist after the loop.

Cumulative product

- This cumulative idea can be used with other operators:

```
int product = 1;
for (int i = 1; i <= 20; i++) {
    product = product * 2;
}
System.out.println("2 ^ 20 = " + product);
```

- How would we make the base and exponent adjustable?

Scanner and cumul. sum

- We can do a cumulative sum of user input:

```
Scanner console = new Scanner(System.in);
int sum = 0;
for (int i = 1; i <= 100; i++) {
    System.out.print("Type a number: ");
    sum = sum + console.nextInt();
}
System.out.println("The sum is " + sum);
```

From Lab: Write a method `countFactors` that returns the number of factors of an integer.

i.e. `countFactors(24)` returns 8 because

1, 2, 3, 4, 6, 8, 12, and 24 are factors of 24