

Building Java Programs

Chapter 4

Lecture 4-2: Advanced `if/else`; Cumulative sum;

reading: 4.2, 4.4 - 4.5

NEVER HAVE I FELT SO
CLOSE TO ANOTHER SOUL
AND YET SO HELPLESSLY ALONE
AS WHEN I GOOGLE AN ERROR
AND THERE'S ONE RESULT
A THREAD BY SOMEONE
WITH THE SAME PROBLEM
AND NO ANSWER
LAST POSTED TO IN 2003



WHO WERE YOU,
DENVERCODER?

WHAT DID YOU SEE?!

Advanced `if/else`

reading: 4.4 - 4.5

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;
```

The "dangling if" problem

- What can be improved about the following code?

```
if (x < 0) {  
    System.out.println("x is negative");  
} else if (x >= 0) {  
    System.out.println("x is non-negative");  
}
```

- The second if test is unnecessary and can be removed:

```
if (x < 0) {  
    System.out.println("x is negative");  
} else {  
    System.out.println("x is non-negative");  
}
```

- This is also relevant in methods that use `if` with `return...`

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.

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 logical expressions

- Relational operators have lower precedence than math; logical operators have lower precedence than relational operators

```
5 * 7 >= 3 + 5 * (7 - 1) && 7 <= 11
```

```
5 * 7 >= 3 + 5 * 6 && 7 <= 11
```

```
35 >= 3 + 30 && 7 <= 11
```

```
35 >= 33 && 7 <= 11
```

```
true && true
```

```
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
```

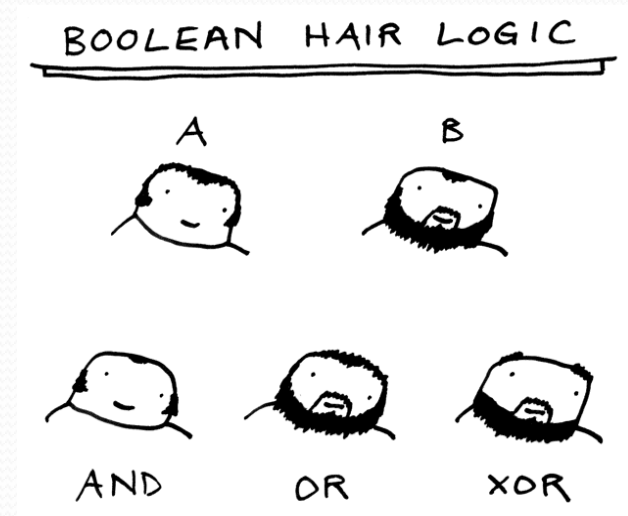
Logical questions

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

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

- $y < x \ \&\& \ y \leq z$
- $x \% 2 == y \% 2 \ || \ x \% 2 == z \% 2$
- $x \leq y + z \ \&\& \ x \geq y + z$
- $!(x < y \ \&\& \ x < z)$
- $(x + y) \% 2 == 0 \ || \ !((z - y) \% 2 == 0)$

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



Cumulative algorithms

reading: 4.2

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 cumulative 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);
```

Cumulative sum question

- Modify the `Receipt` program from Ch. 2.
 - Prompt for how many people, and each person's dinner cost.
 - Use static methods to structure the solution.
- Example log of execution:

How many people ate? 4

Person #1: How much did your dinner cost? 20.00

Person #2: How much did your dinner cost? 15

Person #3: How much did your dinner cost? 30.0

Person #4: How much did your dinner cost? 10.00

Subtotal: \$75.0

Tax: \$6.0

Tip: \$11.25

Total: \$92.25

Cumulative sum answer

```
// This program enhances our Receipt program using a cumulative sum.
```

```
import java.util.*;
```

```
public class Receipt2 {
```

```
    public static void main(String[] args) {
```

```
        Scanner console = new Scanner(System.in);
```

```
        double subtotal = meals(console);
```

```
        results(subtotal);
```

```
    }
```

```
// Prompts for number of people and returns total meal subtotal.
```

```
public static double meals(Scanner console) {
```

```
    System.out.print("How many people ate? ");
```

```
    int people = console.nextInt();
```

```
    double subtotal = 0.0;           // cumulative sum
```

```
    for (int i = 1; i <= people; i++) {
```

```
        System.out.print("Person #" + i +
```

```
                           ": How much did your dinner cost? ");
```

```
        double personCost = console.nextDouble();
```

```
        subtotal = subtotal + personCost; // add to sum
```

```
    }
```

```
    return subtotal;
```

```
}
```

```
...
```

Cumulative answer, cont'd.

...

```
// Calculates total owed, assuming 8% tax and 15% tip
```

```
public static void results(double subtotal) {  
    double tax = subtotal * .08;  
    double tip = subtotal * .15;  
    double total = subtotal + tax + tip;  
  
    System.out.println("Subtotal: $" + subtotal);  
    System.out.println("Tax: $" + tax);  
    System.out.println("Tip: $" + tip);  
    System.out.println("Total: $" + total);  
}  
}
```


if/else, return question

- Write a method `countFactors` that returns the number of factors of an integer.
 - `countFactors(24)` returns 8 because 1, 2, 3, 4, 6, 8, 12, and 24 are factors of 24.

- Solution:

```
// Returns how many factors the given number has.
```

```
public static int countFactors(int number) {  
    int count = 0;  
    for (int i = 1; i <= number; i++) {  
        if (number % i == 0) {  
            count++; // i is a factor of number  
        }  
    }  
    return count;  
}
```