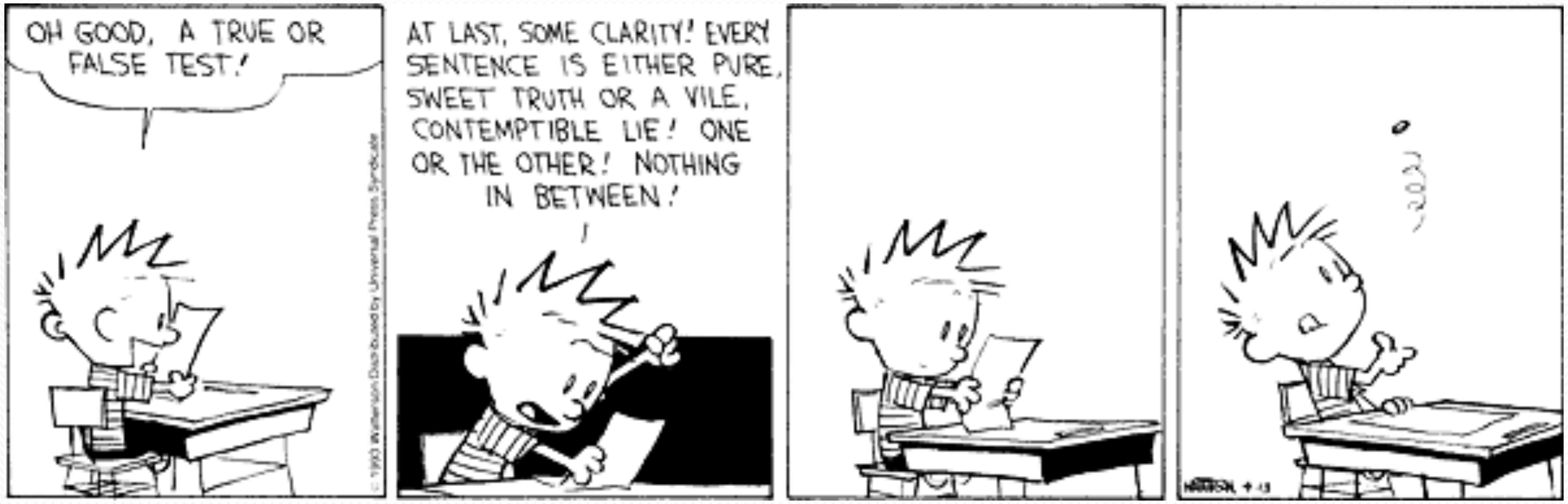
# Building Java Programs

Chapter 5
Lecture 12: boolean Logic

**reading: 5.3**


OH GOOD, A TRUE OR FALSE TEST!
AT LAST, SOME CLARITY! EVERY SENTENCE IS EITHER PURE, SWEET TRUTH OR A VILE, CONTEMPTIBLE LIE! ONE OR THE OTHER! NOTHING IN BETWEEN!
© 1993 Watterson Distributed by Universal Press Syndicate
WATTERSON 4-13

# Type `boolean`

- **`boolean`**: A logical type whose values are `true` and `false`.
  - A logical ***test*** is actually a `boolean` expression.
  - Like other types, it is legal to:
    - create a `boolean` variable
    - pass a `boolean` value as a parameter
    - return a `boolean` value from methods
    - call a method that returns a `boolean` and use it as a test

```
boolean minor    = age < 21;
boolean isProf   = name.contains("Prof");
boolean lovesCSE = true;

// allow only CSE-loving students over 21
if (minor || isProf || !lovesCSE) {
    System.out.println("Can't enter the club!");
}
```

## Using `boolean`

- Why is type `boolean` useful?
  - Can capture a complex logical test result and use it later
  - Can write a method that does a complex test and returns it
  - Makes code more readable
  - Can pass around the result of a logical test (as param/return)

```
boolean goodAge    = age >= 19 && age < 29;
boolean goodHeight = height >= 78 && height < 84;
boolean rich       = salary >= 100000.0;

if ((goodAge && goodHeight) || rich) {
    System.out.println("Okay, let's go out!");
} else {
    System.out.println("It's not you, it's me...");
}
```

# Returning `boolean`

```
public static boolean isPrime(int n) {
    int factors = 0;
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) {
            factors++;
        }
    }
    if (factors == 2) {
        return true;
    } else {
        return false;
    }
}
```

- Calls to methods returning `boolean` can be used as tests:

```
if (isPrime(57)) {
    ...
}
```

## "Boolean Zen", part 1

- Students new to `boolean` often test if a result is `true`:

```
if (isPrime(57) == true) {      // bad
    ...
}
```

- But this is unnecessary and redundant.  Preferred:

```
if (isPrime(57)) {              // good
    ...
}
```

- A similar pattern can be used for a `false` test:

```
if (isPrime(57) == false) {     // bad
if (!isPrime(57)) {             // good
```

## "Boolean Zen", part 2

- Methods that return `boolean` often have an `if/else` that returns `true` or `false`:

```
public static boolean bothOdd(int n1, int n2) {
    if (n1 % 2 != 0 && n2 % 2 != 0) {
        return true;
    } else {
        return false;
    }
}
```

  - But the code above is unnecessarily verbose.

## Solution w/ boolean variable

- We could store the result of the logical test.

```
public static boolean bothOdd(int n1, int n2) {
    boolean test = (n1 % 2 != 0 && n2 % 2 != 0);
    if (test) {    // test == true
        return true;
    } else {       // test == false
        return false;
    }
}
```

- Notice: Whatever `test` is, we want to return that.
  - If `test` is `true` , we want to return `true`.
  - If `test` is `false`, we want to return `false`.

# Solution w/ "Boolean Zen"

- Observation: The `if/else` is unnecessary.
  - The variable `test` stores a `boolean` value;
    its value is exactly what you want to return. So return that!

```
public static boolean bothOdd(int n1, int n2) {
    boolean test = (n1 % 2 != 0 && n2 % 2 != 0);
    return test;
}
```

- An even shorter version:
  - We don't even need the variable `test`.
    We can just perform the test and return its result in one step.

```
public static boolean bothOdd(int n1, int n2) {
    return (n1 % 2 != 0 && n2 % 2 != 0);
}
```

## "Boolean Zen" template

- Replace

```
public static boolean name(parameters) {
    if (test) {
        return true;
    } else {
        return false;
    }
}
```

- with

```
public static boolean name(parameters) {
    return test;
}
```

# Improved `isPrime` method

- The following version utilizes Boolean Zen:

```
public static boolean isPrime(int n) {
    int factors = 0;
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) {
            factors++;
        }
    }
    return factors == 2;   // if n has 2 factors -> true
}
```

# De Morgan's Law

- **De Morgan's Law**: Rules used to negate boolean tests.
  - Useful when you want the opposite of an existing test.

| Original Expression | Negated Expression | Alternative |
|---|---|---|
| `a && b` | `!a \|\| !b` | `!(a && b)` |
| `a \|\| b` | `!a && !b` | `!(a \|\| b)` |

- Example:

| Original Code | Negated Code |
|---|---|
| `if (x == 7 && y > 3) {`<br>`    ...`<br>`}` | `if (x` **`!=`** `7` **`\|\|`** `y` **`<=`** `3) {`<br>`    ...`<br>`}` |

# Boolean practice questions

- Write a method named `isVowel` that returns whether a `String` is a vowel (a, e, i, o, or u), case-insensitively.
  - `isVowel("q")` returns `false`
  - `isVowel("A")` returns `true`
  - `isVowel("e")` returns `true`

- Change the above method into an `isNonVowel` that returns whether a `String` is any character except a vowel.
  - `isNonVowel("q")` returns `true`
  - `isNonVowel("A")` returns `false`
  - `isNonVowel("e")` returns `false`

# Boolean practice answers

```
// Enlightened version.  I have seen the true way (and false way)
public static boolean isVowel(String s) {
    return s.equalsIgnoreCase("a") || s.equalsIgnoreCase("e") ||
           s.equalsIgnoreCase("i") || s.equalsIgnoreCase("o") ||
           s.equalsIgnoreCase("u");
}


// Enlightened "Boolean Zen" version
public static boolean isNonVowel(String s) {
    return !s.equalsIgnoreCase("a") && !s.equalsIgnoreCase("e") &&
           !s.equalsIgnoreCase("i") && !s.equalsIgnoreCase("o") &&
           !s.equalsIgnoreCase("u");

    // or, return !isVowel(s);
}
```

# When to return?

- Methods with loops and return values can be tricky.
  - When and where should the method return its result?

- Write a method `hasVowel` that accepts a `String` parameter and that returns true if the `String` contains at least one vowel. Return false otherwise.

# Flawed solution

```
// Returns true if s contains at least 1 vowel.
public static boolean hasVowel(String s) {
    for (int i = 0; i < s.length(); i++) {
        if (isVowel(s.substring(i, i + 1))) {
            return true;
        } else {
            return false;
        }
    }
}
```

- The method always returns immediately after the first letter!
- If the first letter is not a vowel but the rest of the word contains a vowel, the result is wrong.

## Returning at the right time

```
// Returns true if s contains at least 1 vowel.
public static boolean hasVowel(String s) {
    for (int i = 0; i < s.length(); i++) {
        // found vowel - exit
        if (isVowel(s.substring(i, i + 1))) {
            return true;
        }
    }
    return false;   // if we get here, there was no vowel
}
```

- Returns `true` immediately if vowel is found.
- If vowel isn't found, the loop continues walking the string.
- If no character is a vowel, the loop ends and we return `false`.