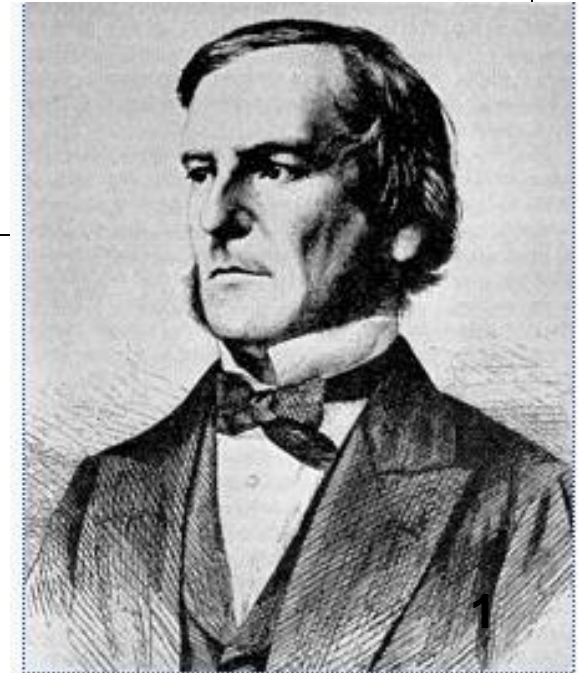


# Topic 16

## boolean logic

"No matter how correct a mathematical theorem may appear to be, one ought never to be satisfied that there was not something imperfect about it until it also gives the impression of being beautiful."

- George Boole



# 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 < 18;  
boolean isProf    = name.contains("Prof");  
boolean lovesCS   = true;
```

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

# Using boolean

## ► Why is type `boolean` useful?

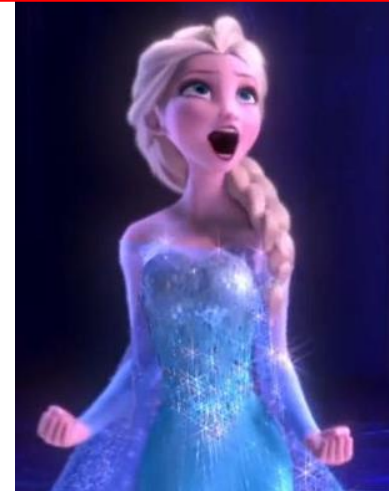
- Can capture a complex logical test result and use it later
- Can write a method that does a complex test and returns it
- Makes code more readable
- Can pass around the result of a logical test (as param/return)

```
boolean goodTemp      = 50 <= temp && temp <= 90;
boolean goodHumidity = humidity <= 70;
boolean haveTime      = time >= 90; // minutes
if ((goodTemp && goodHumidity) || haveTime) {
    System.out.println("Let's RIDE BIKES!!!!");
} else {
    System.out.println("Maybe tomorrow");
}
```

# Returning boolean

```
public static boolean isPrime(int n) {  
    int factors = 0;  
    for (int i = 1; i <= n; i++) {  
        if (n % i == 0) {  
            factors++;  
        }  
    }  
    // NOTE: GACKY STYLE AHEAD!! GACKY == BAD!!  
    if (factors == 2) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

I CAN'T !!!!!!!



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

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

# Boolean question

- ▶ Improve our "rhyme" / "alliterate" program to use boolean methods to test for rhyming and alliteration.

Type two words: **Bare blare**

They rhyme!

They alliterate!

# Boolean answer

```
if (rhyme(word1, word2)) {  
    System.out.println("They rhyme!");  
}  
if (alliterate(word1, word2)) {  
    System.out.println("They alliterate!");  
}  
...
```

```
// Returns true if s1 and s2 end with the same two letters.
```

```
// NOTE: GACKY STYLE AHEAD!!
```

```
public static boolean rhyme(String s1, String s2) {  
    if (s2.length() >= 2 && s1.endsWith(s2.substring(s2.length() - 2))) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

```
// Returns true if s1 and s2 start with the same letter.
```

```
// NOTE: GACKY STYLE AHEAD!!
```

```
public static boolean alliterate(String s1, String s2) {  
    if (s1.startsWith(s2.substring(0, 1))) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

# "Boolean Zen", part 2

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

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

- ▶ But this is unnecessary and redundant. Preferred:

```
if (isPrime(57)) {          // elegant, zen
    ...
}
```

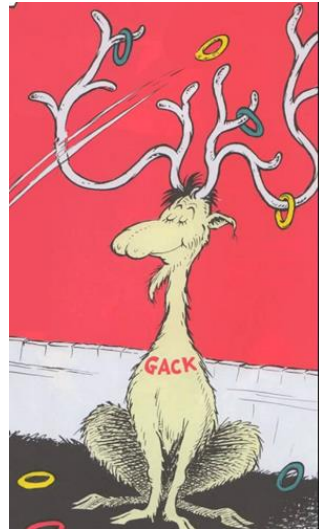
- ▶ A similar pattern can be used for a `false` test:

```
if (isPrime(57) == false) { // inelegant
if (!isPrime(57)) {        // elegant, zen
```

# "Boolean Zen", part 2

► Programmers often write methods that return boolean often have an `if/else` that returns `true` or `false`:

```
// NOTE: GACKY STYLE AHEAD!!  
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);  
    // NOTE: BAD STYLE AHEAD!!  
    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  
}
```

- ▶ Modify the Rhyme program to use Boolean Zen.

# Boolean Zen answer

```
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    System.out.print("Type two words: ");
    String word1 = console.next().toLowerCase();
    String word2 = console.next().toLowerCase();

    if (rhyme(word1, word2)) {
        System.out.println("They rhyme!");
    }
    if (alliterate(word1, word2)) {
        System.out.println("They alliterate!");
    }
}

// Returns true if s1 and s2 end with the same two letters.
public static boolean rhyme(String s1, String s2) {
    return s2.length() >= 2 && s1.endsWith(s2.substring(s2.length() - 2));
}

// Returns true if s1 and s2 start with the same letter.
public static boolean alliterate(String s1, String s2) {
    return s1.startsWith(s2.substring(0, 1));
}
```

# De Morgan's Law

- ▶ **De Morgan's Law:** Rules used to negate boolean tests.
- ▶  $!(a \ \&\& \ b) == !a \ || \ !b$
- ▶  $!(a \ || \ b) == !a \ \&\& \ !b$ 
  - Useful when you want the opposite of an existing test.

| Original Expression         | Negated Expression            | Alternative                    |
|-----------------------------|-------------------------------|--------------------------------|
| <code>a &amp;&amp; b</code> | <code>!a    !b</code>         | <code>!(a &amp;&amp; b)</code> |
| <code>a    b</code>         | <code>!a &amp;&amp; !b</code> | <code>!(a    b)</code>         |

– Example:

| Original Code  | Negated Code   |
|--|--|
| <pre>if (x == 7 &amp;&amp; y &gt; 3) {<br/>    ...<br/>}</pre> | <pre>if (x <b>!=</b> 7 <b>  </b> y <b>&lt;=</b> 3) {<br/>    ...<br/>}</pre> |

# Clicker 1

▶ Which of the following is equivalent to the boolean expression?  $x$ ,  $y$ , and  $z$ 's are ints

$! ( (x \geq y) \ || \ (z \neq x) )$

A.  $! (x \geq y) \ \&\& \ ! (z \neq x)$

B.  $! (x \geq y) \ || \ ! (z \neq x)$

C.  $(x == y) \ \&\& \ (z \geq x)$

D.  $(x < y) \ \&\& \ (z == x)$

E. More than one of A - D is correct

# 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`
- ▶ Write a method `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 `seven` that accepts a `Random` parameter and uses it to draw up to ten lotto numbers from 1-30.
  - If any of the numbers is a lucky 7, the method should stop and return `true`. If none of the ten are 7 it should return `false`.
  - The method should print each number as it is drawn.

15 29 18 29 11 3 30 17 19 22 (first call)

29 5 29 4 **7** (second call)

# Flawed solution

```
// Draws 10 lotto numbers; returns true if one is 7.
public static boolean seven(Random rand) {
    for (int i = 1; i <= 10; i++) {
        int num = rand.nextInt(30) + 1;
        System.out.print(num + " ");

        if (num == 7) {
            return true;
        } else {
            return false;
        }
    }
}
```

- The method always returns immediately after the first roll.
- This is wrong if that draw isn't a 7; we need to keep drawing.

# Returning at the right time

```
// Draws 10 lotto numbers; returns true if one is 7.
public static boolean seven(Random rand) {
    for (int i = 1; i <= 10; i++) {
        int num = rand.nextInt(30) + 1;
        System.out.print(num + " ");

        if (num == 7) { // found lucky 7; can exit now
            return true;
        }
    }

    return false; // if we get here, there was no 7
}
```

- ▶ Returns `true` immediately if 7 is found.
- ▶ If 7 isn't found, the loop continues drawing lotto numbers.
- ▶ If all ten aren't 7, the loop ends and we return `false`.

# Boolean return questions

- ▶ `hasAnOddDigit` : returns `true` if any digit of an integer is odd.
  - `hasAnOddDigit(4822116)` returns `true`
  - `hasAnOddDigit(2448)` returns `false`
- ▶ `allDigitsOdd` : returns `true` if every digit of an integer is odd.
  - `allDigitsOdd(135319)` returns `true`
  - `allDigitsOdd(9174529)` returns `false`
- ▶ `isAllVowels` : returns `true` if every char in a `String` is a vowel.
  - `isAllVowels("eIeIo")` returns `true`
  - `isAllVowels("oink")` returns `false`
    - These problems are available in our Practice-It! system under **5.x**.

# Boolean return answers

```
public static boolean hasAnOddDigit(int n) {
    while (n != 0) {
        if (n % 2 != 0) {    // check whether last digit is odd
            return true;
        }
        n = n / 10;
    }
    return false;
}

public static boolean allDigitsOdd(int n) {
    while (n != 0) {
        if (n % 2 == 0) {    // check whether last digit is even
            return false;
        }
        n = n / 10;
    }
    return true;
}

public static boolean isAllVowels(String s) {
    for (int i = 0; i < s.length(); i++) {
        String letter = s.substring(i, i + 1);
        if (!isVowel(letter)) {
            return false;
        }
    }
    return true;
}
```

# while loop question

- ▶ Write a method `digitSum` that accepts an integer parameter and returns the sum of its digits.
  - Assume that the number is non-negative.
  - Example: `digitSum(29107)` returns `2+9+1+0+7` or `19`
  - Hint: Use the `%` operator to extract a digit from a number.

# while loop answer

```
public static int digitSum(int n) {  
    n = Math.abs(n); // handle negatives  
  
    int sum = 0;  
    while (n > 0) {  
        // add last digit  
        sum = sum + (n % 10);  
        // remove last digit  
        n = n / 10;  
    }  
  
    return sum;  
}
```