

# Building Java Programs

## Chapter 5 Lecture 5-3: Boolean Logic

**reading: 5.2**

self-check: #11 - 17

exercises: #12

videos: Ch. 5 #2

# Type boolean

- **boolean**: A logical type whose values are `true` and `false`.
  - A **test** in an `if`, `for`, or `while` is a boolean expression.
  - You can create boolean variables, pass boolean parameters, return boolean values from methods, ...

```
boolean minor = (age < 21);
boolean expensive = iPhonePrice > 200.00;
boolean iLoveCS = true;

if (minor) {
    System.out.println("Can't purchase alcohol!");
}

if (iLoveCS || !expensive) {
    System.out.println("Buying an iPhone");
}
```



# Methods that return boolean

- Methods can return boolean values.
  - A call to such a method can be a loop or **if test**.

```
Scanner console = new Scanner(System.in);  
System.out.print("Type your name: ");  
String line = console.nextLine();
```

```
if (line.startsWith("Dr.")) {  
    System.out.println("Will you marry me?");  
} else if (line.endsWith(", Esq.")) {  
    System.out.println("And I am Ted 'Theodore' Logan!");  
}
```

# De Morgan's Law

- **De Morgan's Law:**

Rules used to *negate* or *reverse* boolean expressions.

- Useful when you want the opposite of a known boolean 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) {     ... }</pre>	<pre>if (x != 7    y &lt;= 3) {     ... }</pre>



# "Boolean Zen", part 1

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

```
if (bothOdd(7, 13) == true) {    // bad
    ...
}
```

- But this is unnecessary and redundant. Preferred:

```
if (bothOdd(7, 13)) {           // good
    ...
}
```

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

```
if (bothOdd(7, 13) == false) {  // bad
if (!bothOdd(7, 13)) {         // 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

- One option is to 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;  
}
```

# Boolean - Rhyme/Alliterate

Write a program that prompts the user for two words and reports whether they "rhyme" (end with the same last two letters) and/or "alliterate" (start with the same letter).

Must match format of expected output (website, 5 example runs).

Hints:

- Create rhyme() & alliterate()
- Call rhyme(), alliterate() in main
- No print in rhyme() / alliterate()
- All printing in main()
- Scanner next()
- String endsWith(), substring()
- Boolean && and ! in main()
- Boolean && operator in rhyme()
- No ifs in rhyme() or alliterate()

(run #1)

Type two words: car STAR

They rhyme!

(run #2)

Type two words: Bare blare

They rhyme!

They alliterate (start with the same letter)!

(run #3)

Type two words: booyah socks

They have nothing in common.

(run #4)

Type two words: a a

They alliterate (start with the same letter)!

(run #5)

Type two words: aa aa

They rhyme!

They alliterate (start with the same letter)!



# 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 (a, e, i, o, or u).
  - `isNonVowel("q")` returns `true`
  - `isNonVowel("A")` returns `false`
  - `isNonVowel("e")` returns `false`
- Write methods named `allVowels` and `containsVowel`.

# Boolean practice answers

```
public static boolean isVowel(String s) {  
    if (s.equalsIgnoreCase("a") || s.equalsIgnoreCase("e") ||  
        s.equalsIgnoreCase("i") || s.equalsIgnoreCase("o") ||  
        s.equalsIgnoreCase("u")) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

```
public static boolean isNonVowel(String s) {  
    if (!s.equalsIgnoreCase("a") && !s.equalsIgnoreCase("e") &&  
        !s.equalsIgnoreCase("i") && !s.equalsIgnoreCase("o") &&  
        !s.equalsIgnoreCase("u")) {  
        return true;  
    } else {  
        return false;  
    }  
}
```



# Boolean practice answers 2

**// 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 version**

```
public static boolean isNonVowel(String s) {  
    return !s.equalsIgnoreCase("a") && !s.equalsIgnoreCase("e") &&  
           !s.equalsIgnoreCase("i") && !s.equalsIgnoreCase("o") &&  
           !s.equalsIgnoreCase("u");  
}
```

# When to return?

- In methods that involve a loop and a `boolean` return:
  - How do you figure out whether to return `true` or `false`?
  - When should the method return its result?
- Example problem:
  - Write a method `seven` that accepts a `Random` parameter and uses it to pick up to 10 lotto numbers between 1 and 30.
  - The method should print each number as it is drawn.
    - Example output from 2 calls:  
15 29 18 29 11 3 30 17 19 22  
29 5 29 16 4 **7**
  - If any of the numbers is a lucky 7, the method should return `true`. Otherwise, it should return `false`.



# Flawed solution

- Common incorrect solution:

```
// Draws 10 random lotto numbers.  
// Returns true if one of them is a lucky 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 tries to return immediately after the first roll.
- This is bad, if that roll isn't a 7; we need to roll all 10 times to see if any of them is a 7.

# Returning at the right time

- Corrected code:

```
// Draws 10 random lotto numbers.  
// Returns true if one of them is a lucky 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;  
        }  
    }  
  
    // if we get here, we know there was no 7  
    return false;  
}
```

- Returns immediately if 7 is found, because the answer must be `true`. If 7 isn't found, we draw the next lotto number. If all 10 aren't 7, the loop ends and we return `false`.



# Boolean return questions

- Write a method named `hasAnOddDigit` that returns whether any digit of a positive integer is odd.
  - `hasAnOddDigit(4822116)` returns `true`
  - `hasAnOddDigit(2448)` returns `false`
- Write a method named `allDigitsOdd` that returns whether every digit of a positive integer is odd.
  - `allDigitsOdd(135319)` returns `true`
  - `allDigitsOdd(9175293)` returns `false`
- Write a method named `isAllVowels` that returns `true` if every character in a `String` is a vowel, else `false`.
  - `isAllVowels("eIeIo")` returns `true`
  - `isAllVowels("oink")` returns `false`

# 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;
}
```