Type `boolean`

Subset of the Supplement Lesson slides from: Building Java Programs, Chapter 5.3
by Stuart Reges and Marty Stepp (http://www.buildingjavaprograms.com/)

Methods that are tests

- Some methods return logical values.
  - A call to such a method is used as a **test** in a loop or `if`.

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

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

`String` test methods

Method	Description
`equals(`**str**`)`	whether two strings contain the same characters
`equalsIgnoreCase(`**str**`)`	whether two strings contain the same characters, ignoring upper vs. lower case
`startsWith(`**str**`)`	whether one contains other's characters at start
`endsWith(`**str**`)`	whether one contains other's characters at end
`contains(`**str**`)`	whether the given string is found within this one

```
String name = console.next();
if (name.contains("Prof")) {
    System.out.println("When are your office hours?");
} else if (name.equalsIgnoreCase("STUART")) {
    System.out.println("Let's talk about meta!");
}
```

Type `boolean`

- **`boolean`**: A logical type whose values are `true` and `false`.
  - A logical **test** is actually a `boolean` expression.
  - 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 >= 12 && 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` var

- 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
}
```

"Short-circuit" evaluation

- Java stops evaluating a test if it knows the answer.
  - `&&` stops early if any part of the test is `false`
  - `||` stops early if any part of the test is `true`

- The following test will crash if x is greater than s1's length:

```
// if test for the character at x is an "s".
if ( s1.substring(x-1,x).equals("s") && x < s1.length() ){
   ... }
```

- The following test will not crash; it stops after `s1.length() < x` :

```
// if test for the character at x is an "s".
if ( x < s1.length() && s1.substring(x-1,x).equals("s") ){
   ... }
```

"Short-circuit" evaluation 2

- Java stops evaluating a test if it knows the answer.
  - `&&` stops early if any part of the test is `false`
  - `||` stops early if any part of the test is `true`

- The following test will crash if s2's length is less than 2:

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

- The following test will not crash; it stops if length < 2:

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

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) {`
`  ...`
`}`
	`if (x != 7 || y <= 3) {`
	`  ...`
	`}`

Truth Tables:

p	q	p && q	!(p && q)	!p	!q	!p || !q
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

p	q	p || q	!(p || q)	!p	!q	!p && !q
T	T	T	F	F	F	F
T	F	T	F	F	T	F
F	T	T	F	T	F	F
F	F	F	T	T	T	T

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 `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 roll isn't a 7; we need to keep rolling.

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`.

Another: Example... `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) {
        sum = sum + (n % 10);       // add last digit
        n = n / 10;                 // remove last digit
    }

    return sum;
}
```

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;
}
```