Building Java Programs

Chapter 5: Program Logic and Indefinite Loops

Lecture outline

- more boolean logic
  - De Morgan's Law
  - Methods that return `boolean`
  - "Boolean Zen"

- indefinite loop variations
  - the `do/while` loop
  - the `break` statement

- logical assertions

While loop question

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

While loop answer

- The following code implements the method:

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

"Boolean Zen" and returning boolean values

reading: 5.2, 4.5

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
`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) {`
	`    ...`
	`}`

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`

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 Zen"

- Methods that return `boolean` often have an `if/else`:

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

- There is a better way to write this same code.
  - If you understand the way of "Boolean Zen", you will see a much shorter way to solve this same problem.

Solution w/ boolean variable

- Consider: We could capture the result of the 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` , return `true`.
  - If `test` is `false`, 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 do 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 (<condition>) {
        return true;
    } else {
        return false;
    }
}
```

- with:

```
public static boolean <name>(<parameters>) {
    return <condition>;
}
```

  - Exercise: Modify `isVowel` / `isNonVowel` to use Boolean Zen.

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 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 named `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();
        System.out.print(num + " ");
        if (num == 7) {   // found lucky 7; can exit now
            return true;
        }
    }

    // if we get here, we rolled 10 times and the
    // method never returned, so we know there was no 7
    return false;
}
```

  - The method returns immediately if a 7 is found, because we know right away that the answer must be `true`. If a 7 isn't found, we continue drawing the rest of the 10 lotto numbers. If all 10 aren't 7, the loop ends and we return `false`.

Boolean return questions

- Write a method named `hasAnOddDigit` that returns whether <u>any</u> digit of a positive integer is odd.
  - `hasAnOddDigit(4822`**`11`**`6)` returns `true`
  - `hasAnOddDigit(2448)` returns `false`

- Write a method named `allDigitsOdd` that returns whether <u>every</u> digit of a positive integer is odd.
  - `allDigitsOdd(135319)` returns `true`
  - `allDigitsOdd(9175`**`2`**`93)` returns `false`

- Write a method named `isAllVowels` that returns `true` if <u>every</u> 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;
}
```