Loops

Loops: `while` and `for`

`while` loops can do everything `for` loops can, but `while` loops can also do more (event-driven vs. counter-driven)

For counter-driven loops, a `for` loop is more centrally-controlled and easier to modify, so use those.

Quick Hands-on Example

Write a loop to add all odd numbers between (already declared in prior code) `int x` and `int y`.

Things to think about:

1. What kind of loop should you use?
2. Do you need to add in an `if` statement?
3. Is there a way to formulate this without an `if` statement?

do-while loops

One more type of loop: `do` loops

```
do {
    <statements>
} while (<condition>);
```

`do` loops always execute *at least once*, regardless of the value of the condition.

When to use what?

User input validation - always need to prompt the user for input.

```
boolean goodValue = false;
do {
    System.out.println(‘‘Enter a value for x:’’);
    if (scnr.hasNextInt()) {
        x = scnr.nextInt();
        goodValue = true; // sentinel value
    }
    else {
        System.out.println(‘‘Error: not an int’’);
        scnr.nextLine(); // why?
    }
} while (!goodValue);
```

Using loops to do useful operations!

We began today writing some code to sum all odd numbers in a range. We can do other things with loops, too:

1. Averaging numbers
2. Counting occurrences (e.g. number of 'a's in a String)
3. First/last match (`str.indexOf()` or `str.lastIndexOf()`)
4. Max/min value entered

Nesting loops

Just as we could nest `if` statements, we can put loops inside other loops.

Example: explaining square roots to a four-year-old (extra challenge if we have time: write a program which determines the root using `Random`, outputs the square, and asks the user for the root)