Notes: while loops and fence post problems

definite vs indefinte loop

- **definite loop:** executes a known number of times
  - e.g. Print the numbers 1 to 100 to the screen
- **indefinite loop:** where the number of times the loop will repeat is unknown prior to the code executing
  - e.g. Ask the user for a number until they enter a value between 1 and 100
- for-loops are typically definite loops and while-loops are typically indefinite loops, but this is not always the case. You must pay attention to whether or not you know how many times the loop will number

`while` loop

- Repeatedly executes its body as long as a logical test is true
- Note: The `for` loop is a specialized form of the `while` loop
- Use `while` when it is **unknown** how many times the loop will repeat (meaning you don't know right now, even if you could put in some time and effort to figure out the exact amount)
- Use `for` when it is **known** how many times the loop will repeat

Structure

```
while (test) {
   statement(s);
}
```

- while loops are like repeating if statements; the body of the loop repeats if the condition is true when checked
- the condition of a while loop is checked at the start of the loop; if this condition is not true to start, the loop will not execute
- after the body of the loop is executed, the condition is checked again; if the condition is true again, the loop executes again
- while loops can be used to count or do other tasks that run a set number of times, but in these cases, generally a definite (for loop) is more appropriate

Sentinels

- **sentinel value:** A value that signals the end of user input
- **sentinel loop**: A loop that repeats until a sentinel value is seen

Example

```
// in the following loop the sentinel value is anything but "yes"
// note: that the sentinel value is the stopping case
// and is opposite of the repeating case (the condition)

String again = "yes";
while (again.equals("yes")) {
   // do something

   System.out.print("Go again? (yes/no) > ");
   again = console.next();
}
```

Fencepost problems

- Fencepost problems are when you have a repeating pattern that needs to happen, but part of the pattern doesn't repeat exactly.
  - e.g. If you want to print 1, 2, 3, 4, 5 to the screen using a loop, you need to print each of the numbers followed by a coma, except the last number (5 in this case)
- Sometimes this is illustrated as `|-|-|-|-|` with the `|` representing a fence post and the `-` representing the wire of the fence; Also sometimes called a "loop-and-a-half"
- The idea is that you need 1 more post than you do wire sections; you begin with a post and end with a post
- Common solutions usually have the loop run one less times than needed and then handle the last post outside of the loop

Examples

```
// handles the first post outside the loop
// prints a comma separated list of numbers from 1 up to max
public static void printNumbers(int max) {
   System.out.print(1);
   for(int i = 2; i <= max; i++) {
      System.out.print(", " + i );
   }
   System.out.println();
}
```

```
// handles the last post outside the loop
// prints a comma separated list of numbers from 1 up to max
public static void printNumbers(int max) {
   for(int i = 1; i <= max - 1; i++) {
      System.out.print(i + ", ");
   }
   System.out.println(max);
}
```

Notes: Random numbers and Assertions

Random class

- A Random object generates pseudo-random numbers
- pseudo-random means simulated randomness, but not truly random
- In order to use the Random class you will need to import the util package: import java.util.Random;

Random methods

Method Name	Description
nextInt()	returns a random integer
nextInt(max)	returns a random integer in the range [0, max) (i.e., 0 to max - 1 inclusive)
nextDouble()	returns a random real number in the range [0.0, 1.0)

To get a number in an inclusive range of min to max

```
nextInt(max - min + 1) + min
```

Example code

```
Random rand = new Random();

// randomNumber1 will store a random number in the range 0 – 9
int randomNumber1 = rand.nextInt(10);

// randomNumber2 will store a random number in the range 1 - 20
int randomNumber2 = rand.nextInt(20) + 1;

// randomNumber3 will store a random number that is one of the first 5 even numbers (0, 2, 4, 6, 8)
```

```
int randomNumber3 = rand.nextInt(5) * 2;
```

Boolean

- `boolean` is a type
- `boolean` variables can hold either `true` or `false`
- Using a `boolean`
  - 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 Zen

- do not test a result against `true`

```
// don't do this
if(result == true) {...}

// do this instead
if(result) {...}
```

- do not test if a condition is `true` and then `return true` as a result, just return the boolean expression itself

```
// don't do this
if(count > 10 == true) {
  return true;
}
// do this instead
return count > 10;
```

- do not create variables to return information that can be returned without a variable

```
// don't do this
boolean result;
if(!word.equals("y")) {
  result = false;
}
return result;

// do this instead
if(!word.equals("y")) {
  return false;
}
```

Logical assertions

- Assertion: A statement that is either `true` or `false`
- Tips for solving assertion problems:
  - Right after a variable is initialized, its value is known
  - At the start of a loop's body, the loop's test must be true
  - After a loop, the loop's test must be false
  - Inside a loop's body, the loop's test may become false
  - Reading from a Scanner, reading from a Random object, or parameter values are unknown and usually result in a "Sometimes" assertion
  - If you are unsure, guess "Sometimes"