Building Java Programs

Chapter 5
Lecture 10: `while` Loops,
Fencepost Loops, and Sentinel Loops

reading: 5.1 – 5.2

`String` methods

Method name	Description
`indexOf(`**`str`**`)`	index where the start of the given string appears in this string (-1 if not found)
`length()`	number of characters in this string
`substring(`**`index1`**`,` **`index2`**`)` or `substring(`**`index1`**`)`	the characters in this string from *index1* (inclusive) to *index2* (exclusive); if *index2* is omitted, grabs till end of string
`toLowerCase()`	a new string with all lowercase letters
`toUpperCase()`	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String starz = "Yeezy & Hova";
System.out.println(starz.length());   // 12
```

String method examples

```
// index      012345678901
String s1 = "Stuart Reges";
String s2 = "Marty Stepp";

System.out.println(s1.length());            // 12
System.out.println(s1.indexOf("e"));        // 8
System.out.println(s1.substring(7, 10));    // "Reg"

String s3 = s2.substring(1, 7);
System.out.println(s3.toLowerCase());       // "arty s"
```

- Given the following string:

```
// index        0123456789012345678901
String book = "Building Java Programs";
```

  - How would you extract the word `"Java"` ?

Modifying strings

- Methods like `substring` and `toLowerCase` build and return a new string, rather than modifying the current string.

```
String s = "Aceyalone";
s.toUpperCase();
System.out.println(s);    // Aceyalone
```

- To modify a variable's value, you must reassign it:

```
String s = "Aceyalone";
s = s.toUpperCase();
System.out.println(s);    // ACEYALONE
```

Strings as user input

- `Scanner`'s `next` method reads a word of input as a `String`.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
name = name.toUpperCase();
System.out.println(name + " has " + name.length() +
    " letters and starts with " + name.substring(0, 1));
```

Output:

```
What is your name? Nas
NAS has 3 letters and starts with N
```

- The `nextLine` method reads a line of input as a `String`.

```
System.out.print("What is your address? ");
String address = console.nextLine();
```

Strings question

- Write a program that reads two people's first names and suggests a name for their child

Example Output:

```
Parent 1 first name? **Danielle**
Parent 2 first name? **John**
Child Gender? **f**
Suggested baby name: JODANI

Parent 1 first name? **Danielle**
Parent 2 first name? **John**
Child Gender? **Male**
Suggested baby name: DANIJO
```

The `equals` method

- Objects are compared using a method named `equals`.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Lance")) {
    System.out.println("Pain is temporary.");
    System.out.println("Quitting lasts forever.");
}
```

- Technically this is a method that returns a value of type `boolean`, the type used in logical tests.

`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.endsWith("Kweli")) {
   System.out.println("Pay attention, you gotta listen to hear.");
} else if(name.equalsIgnoreCase("NaS")) {
   System.out.println("I never sleep 'cause sleep is the cousin of
                     death.");
}
```

Type `char`

- `char` : A primitive type representing single characters.
  - Each character inside a `String` is stored as a `char` value.
  - Literal `char` values are surrounded with apostrophe (single-quote) marks, such as `'a'` or `'4'` or `'\n'` or `'\''`
  - It is legal to have variables, parameters, returns of type `char`

```
char letter = 'S';
System.out.println(letter);              // S
```

- `char` values can be concatenated with strings.

```
char initial = 'P';
System.out.println(initial + " Diddy");  // P Diddy
```

The `charAt` method

- The `char`s in a `String` can be accessed using the `charAt` method.

```
String food = "cookie";
char firstLetter = food.charAt(0);    // 'c'

System.out.println(firstLetter + " is for " + food);
System.out.println("That's good enough for me!");
```

- You can use a `for` loop to print or examine each character.

```
String major = "CSE";
for (int i = 0; i < major.length(); i++) {
    char c = major.charAt(i);
    System.out.println(c);
}
```

Output:

```
C
S
E
```

`char` vs. `String`

- `"h"` is a `String`
  `'h'` is a `char` (the two behave differently)

- `String` is an object; it contains methods

```
String s = "h";
s = s.toUpperCase();           // 'H'
int len = s.length();          //  1
char first = s.charAt(0);      // 'H'
```

- `char` is primitive; you can't call methods on it

```
char c = 'h';
c = c.toUpperCase();    // ERROR: "cannot be dereferenced"
```

- What is `s + 1` ? What is `c + 1` ?
- What is `s + s` ? What is `c + c` ?

`char` vs. `int`

- All `char` values are assigned numbers internally by the computer, called *ASCII* values.
  - Examples:
    `'A'` is 65, `'B'` is 66, `' '` is 32
    `'a'` is 97, `'b'` is 98, `'*'` is 42
  - Mixing `char` and `int` causes automatic conversion to `int`.
    `'a' + 10` is 107, `'A' + 'A'` is 130
  - To convert an `int` into the equivalent `char`, type-cast it.
    `(char) ('a' + 2)` is `'c'`

Comparing `char` values

- You can compare `char` values with relational operators:

  `'a' < 'b'` and `'X' == 'X'` and `'Q' != 'q'`

  - An example that prints the alphabet:

```
for (char c = 'a'; c <= 'z'; c++) {
    System.out.print(c);
}
```

- You can test the value of a string's character:

```
String word = console.next();
if (word.charAt(word.length() - 1) == 's') {
    System.out.println(word + " is plural.");
}
```

`String`/`char` question

- A *Caesar cipher* is a simple encryption where a message is encoded by shifting each letter by a given amount.
  - e.g. with a shift of 3, A → D, H → K, X → A, and Z → C

- Write a program that reads a message from the user and performs a Caesar cipher on its letters:

```
Your secret message: Brad thinks Angelina is cute
Your secret key: 3
The encoded message: eudg wklqnv dqjholqd lv fxwh
```

Strings answer 1

```
// This program reads a message and a secret key from the user and
// encrypts the message using a Caesar cipher, shifting each letter.

import java.util.*;

public class SecretMessage {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Your secret message: ");
        String message = console.nextLine();
        message = message.toLowerCase();

        System.out.print("Your secret key: ");
        int key = console.nextInt();

        encode(message, key);
    }

    ...
```

Strings answer 2

```
    // This method encodes the given text string using a Caesar
    // cipher, shifting each letter by the given number of places.
    public static void encode(String text, int shift) {
        System.out.print("The encoded message: ");
        for (int i = 0; i < text.length(); i++) {
            char letter = text.charAt(i);

            // shift only letters (leave other characters alone)
            if (letter >= 'a' && letter <= 'z') {
                letter = (char) (letter + shift);

                // may need to wrap around
                if (letter > 'z') {
                    letter = (char) (letter - 26);
                } else if (letter < 'a') {
                    letter = (char) (letter + 26);
                }
            }
            System.out.print(letter);
        }
        System.out.println();
    }
}
```

Methods using `charAt`

- Write a method `printConsonants` that accepts a `String` as a parameter and prints out that `String` with all vowels removed

  For example, the call:

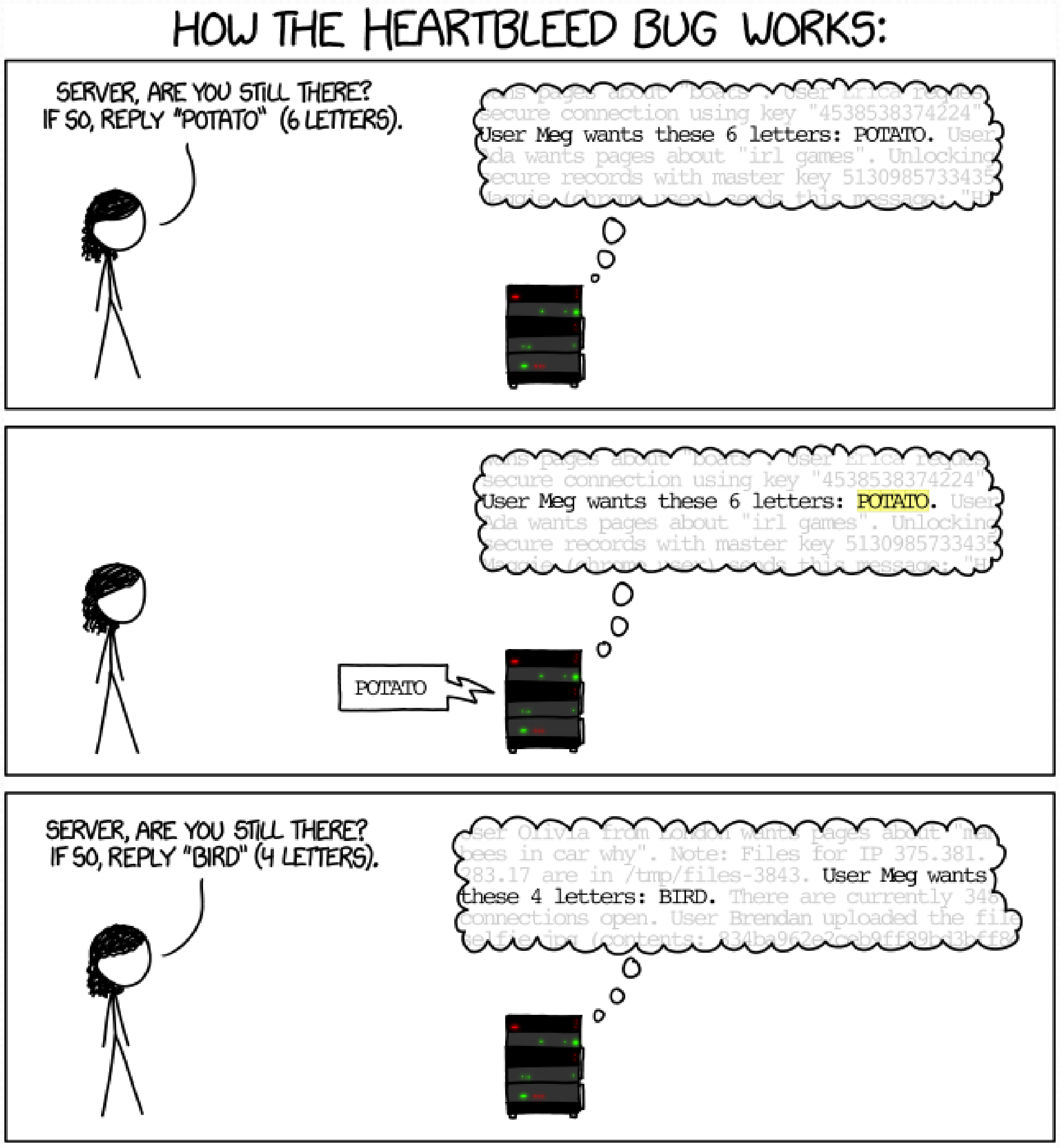
  ```
  printConsonants("atmosphere")
  ```

  should print:

  ```
  tmsphr
  ```

Heartbleed Bug

- OpenSSL
  - Used to encrypt web data
  - Used by Facebook, Google, etc.
  - Written in C
- OpenSSL Heartbeat
  - Make sure connection is still live
  - Send a string and ask for it back
- Bug
  - You can lie to Heartbeat about how long the string is
  - Bug Released March 14th, 2012


HOW THE HEARTBLEED BUG WORKS:
SERVER, ARE YOU STILL THERE? IF SO, REPLY "POTATO" (6 LETTERS).
secure connection using key "4538538374224"
User Meg wants these 6 letters: POTATO. User
Ada wants pages about "irl games". Unlocking
secure records with master key 5130985733435
POTATO
SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).
bees in car why". Note: Files for IP 375.381.
283.17 are in /tmp/files-3843. User Meg wants
these 4 letters: BIRD. There are currently 346
connections open. User Brendan uploaded the file

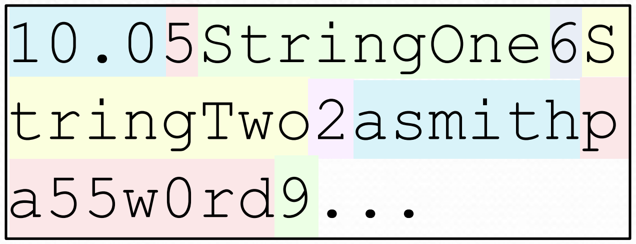
https://xkcd.com/1354/

Heartbleed Bug

Simplified view of computer memory:

```
double y = 10.0;
int w = 5;
String str1 = "StringOne";
int x = 6;
String str2 = "StringTwo";
int y = 2;
String username = "asmith";
String password = "pa55w0rd";
int z = 9;
...
```

Computer Memory



```
10.05StringOne6S
tringTwo2asmithp
a55w0rd9...
```

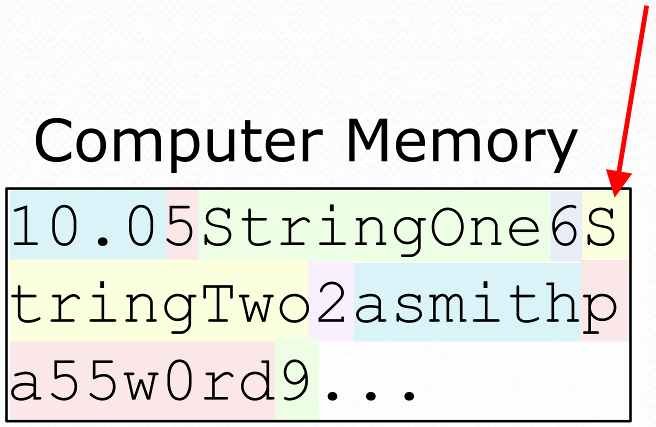
```
str2 length: 9
str2 index 0:
```

Heartbleed Bug

Simplified view of computer memory:

```
double y = 10.0;
int w = 5;
String str1 = “StringOne”;
int x = 6;
String str2 = “StringTwo”;
int y = 2;
String username = “asmith”;
String password = “pa55w0rd”;
int z = 9;
...
```

Computer Memory



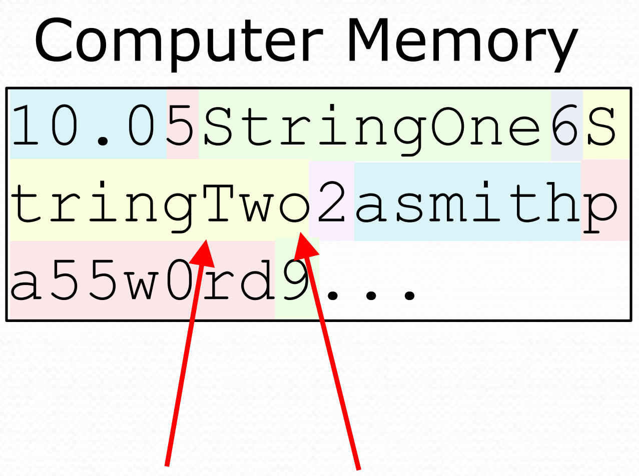
```
10.05StringOne6S
tringTwo2asmithp
a55w0rd9...
```

```
str2 length: 9
str2 index 0: “S”
str2 index 6-8:
```

Heartbleed Bug

Simplified view of computer memory:

```
double y = 10.0;
int w = 5;
String str1 = "StringOne";
int x = 6;
String str2 = "StringTwo";
int y = 2;
String username = "asmith";
String password = "pa55w0rd";
int z = 9;
...
```



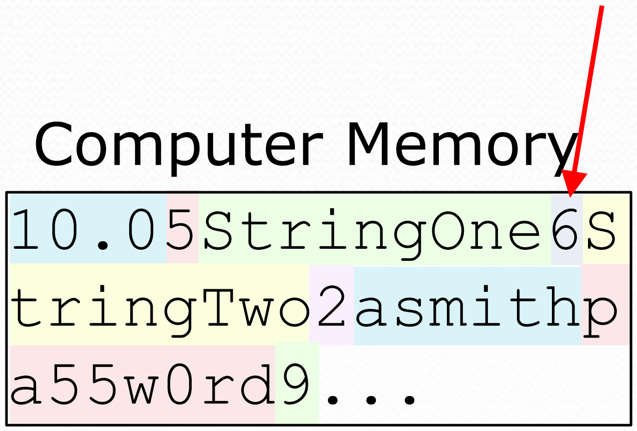
```
str2 length: 9
str2 index 0: "S"
str2 index 6-8: "Two"
str2 index -1:
```

Heartbleed Bug

Simplified view of computer memory:

```
double y = 10.0;
int w = 5;
String str1 = "StringOne";
int x = 6;
String str2 = "StringTwo";
int y = 2;
String username = "asmith";
String password = "pa55w0rd";
int z = 9;
...
```

Computer Memory



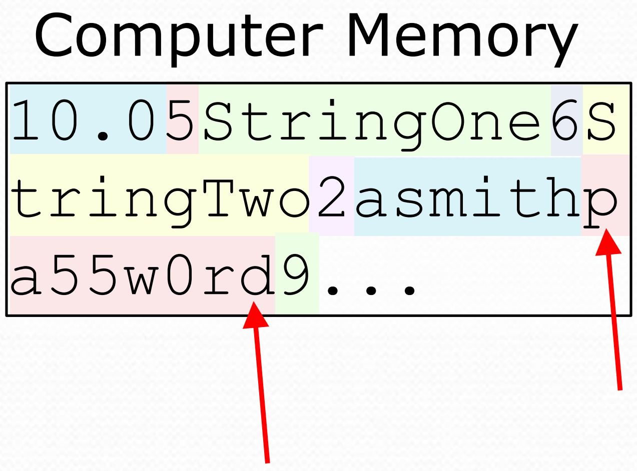
```
10.05StringOne6S
tringTwo2asmithp
a55w0rd9...
```

```
str2 length: 9
str2 index 0: "S"
str2 index 6-8: "Two"
str2 index -1: 6
str2 index 16-23:
```

Heartbleed Bug

Simplified view of computer memory:

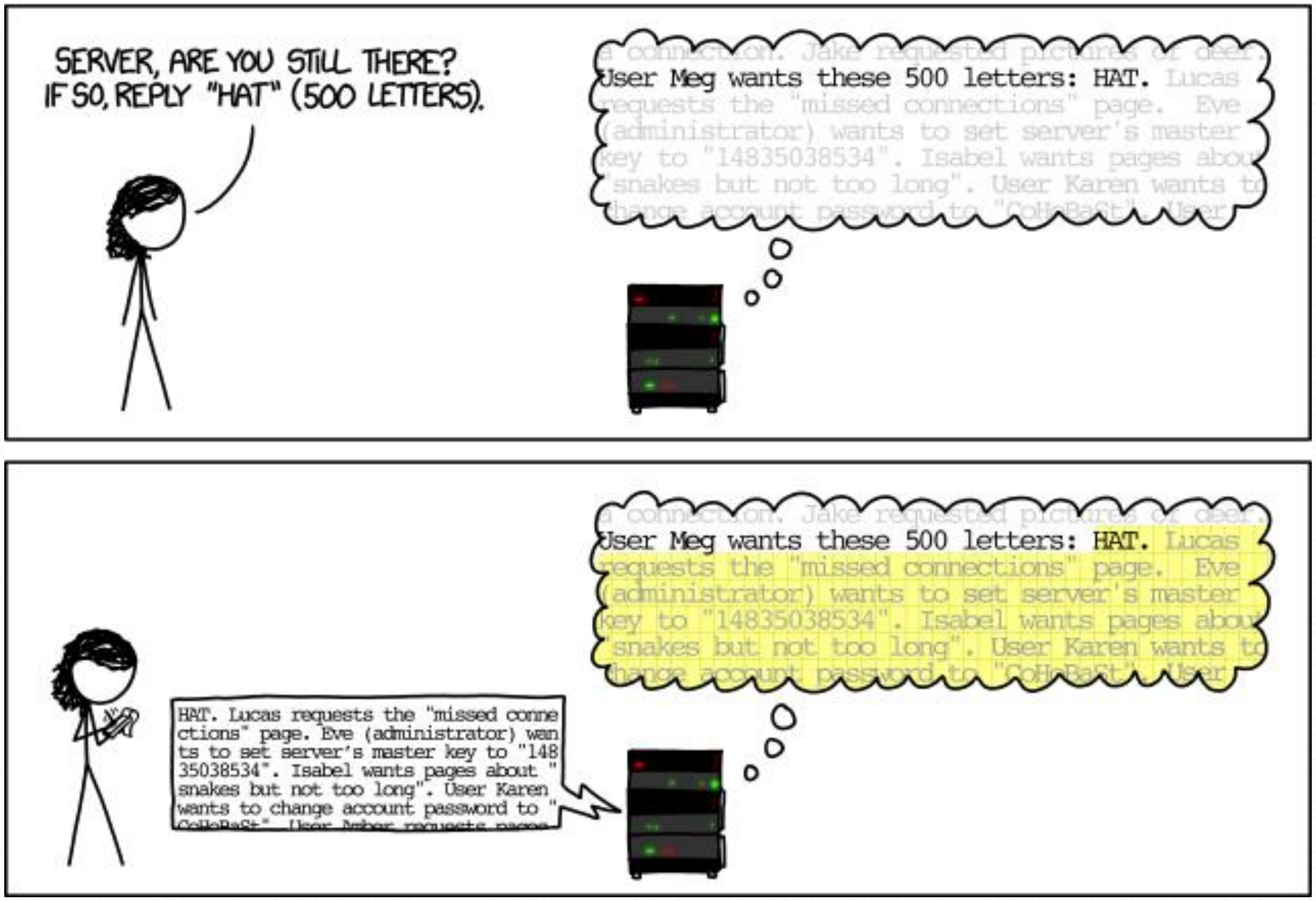
```
double y = 10.0;
int w = 5;
String str1 = "StringOne";
int x = 6;
String str2 = "StringTwo";
int y = 2;
String username = "asmith";
String password = "pa55w0rd";
int z = 9;
...
```



```
str2 length: 9
str2 index 0: "S"
str2 index 6-8: "Two"
str2 index -1: 6
str2 index 16-23: "pa55w0rd"
```

Heartbleed Bug

- Bug
  - Bug Released March 14th, 2012
  - Fix released on April 7th, 2014


SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User
a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User
HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User Amber requests pages

https://xkcd.com/1354/

A deceptive problem...

- Write a method `printLetters` that prints each letter from a word separated by commas.

  For example, the call:

  ```
  printLetters("Atmosphere")
  ```

  should print:

  ```
  A, t, m, o, s, p, h, e, r, e
  ```

Flawed solutions

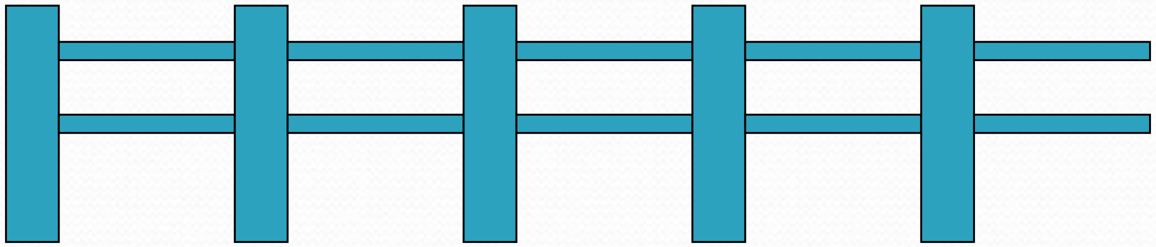
-
  ```
  public static void printLetters(String word) {
          for(int i = 0; i < word.length(); i++) {
              System.out.print(word.charAt(i) + ", ");
          }
          System.out.println();    // end line
  }
  ```
  - Output: `A, t, m, o, s, p, h, e, r, e,`

-
  ```
  public static void printLetters(String word) {
          for(int i = 0; i < word.length(); i++) {
              System.out.print(", " + word.charAt(i));
          }
          System.out.println();    // end line
  }
  ```
  - Output: `, A, t, m, o, s, p, h, e, r, e`

Fence post analogy

- We print *n* letters but need only *n* - 1 commas.
- Similar to building a fence with wires separated by posts:
  - If we use a flawed algorithm that repeatedly places a post + wire, the last post will have an extra dangling wire.

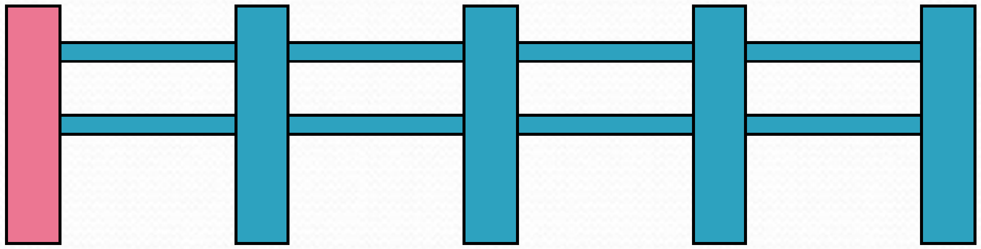
```
for (length of fence) {
    place a post.
    place some wire.
}
```



Fencepost loop

- Add a statement outside the loop to place the initial "post."
  - Also called a *fencepost loop* or a "loop-and-a-half" solution.

```
place a post.
for (length of fence - 1) {
    place some wire.
    place a post.
}
```



Fencepost method solution

-
  ```
  public static void printLetters(String word) {
      System.out.print(word.charAt(0));
      for(int i = 1; i < word.length(); i++) {
          System.out.print(", " + word.charAt(i));
      }
      System.out.println();    // end line
  }
  ```

- Alternate solution: Either first or last "post" can be taken out:

```
public static void printLetters(String word) {
    for(int i = 0; i < word.length() - 1; i++) {
        System.out.print(word.charAt(i) + ", ");
    }
    int last = word.length() - 1;
    System.out.println(word.charAt(last)); // end line
}
```

Fencepost question

- Write a method `printPrimes` that prints all *prime* numbers up to a max.
  - Example: `printPrimes(50)` prints

    ```
    2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47
    ```

  - If the maximum is less than 2, print no output.

- To help you, write a method `countFactors` which returns the number of factors of a given integer.
  - `countFactors(20)` returns `6` due to factors 1, 2, 4, 5, 10, 20.

Fencepost answer

```
// Prints all prime numbers up to the given max.
public static void printPrimes(int max) {
    if (max >= 2) {
        System.out.print("2");
        for (int i = 3; i <= max; i++) {
            if (countFactors(i) == 2) {
                System.out.print(", " + i);
            }
        }
        System.out.println();
    }
}

// Returns how many factors the given number has.
public static int countFactors(int number) {
    int count = 0;
    for (int i = 1; i <= number; i++) {
        if (number % i == 0) {
            count++;    // i is a factor of number
        }
    }
    return count;
}
```

`while` loops

reading: 5.1

Categories of loops

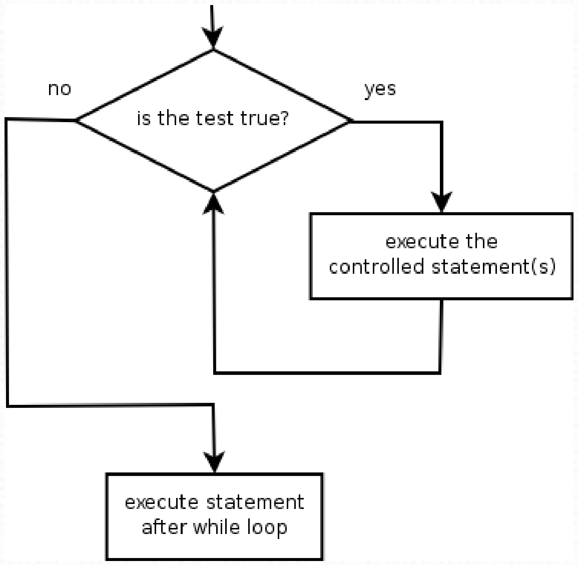
- **definite loop**: Executes a known number of times.
  - The `for` loops we have seen are definite loops.
    - Print "hello" 10 times.
    - Find all the prime numbers up to an integer *n*.
    - Print each odd number between 5 and 127.

- **indefinite loop**: One where the number of times its body repeats is not known in advance.
    - Prompt the user until they type a non-negative number.
    - Print random numbers until a prime number is printed.
    - Repeat until the user has typed "q" to quit.

The `while` loop

- **`while` loop**: Repeatedly executes its body as long as a logical test is true.

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



no
is the test true?
yes
execute the controlled statement(s)
execute statement after while loop

- Example:

```
int num = 1;                          // initialization
while (num <= 200) {                  // test
    System.out.print(num + " ");
    num = num * 2;                    // update
}

// output:  1 2 4 8 16 32 64 128
```

Example `while` loop

```
// finds the first factor of 91, other than 1
int n = 91;
int factor = 2;
while (n % factor != 0) {
    factor++;
}
System.out.println("First factor is " + factor);
// output:  First factor is 7
```

- `while` is better than `for` because we don't know how many times we will need to increment to find the factor.

Sentinel values

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

- Example: Write a program that prompts the user for text until the user types "quit", then output the total number of characters typed.
  - (In this case, "quit" is the sentinel value.)

```
Type a word (or "quit" to exit): hello
Type a word (or "quit" to exit): yay
Type a word (or "quit" to exit): quit
You typed a total of 8 characters.
```

Solution?

```
Scanner console = new Scanner(System.in);
int sum = 0;
String response = "dummy"; // "dummy" value, anything but "quit"

while (!response.equals("quit")) {
    System.out.print("Type a word (or \"quit\" to exit): ");
    response = console.next();
    sum += response.length();
}

System.out.println("You typed a total of " + sum + " characters.");
```

- This solution produces the wrong output. Why?

```
You typed a total of 12 characters.
```

The problem with our code

- Our code uses a pattern like this:

```
sum = 0.
while (input is not the sentinel) {
    prompt for input; read input.
    add input length to the sum.
}
```

- On the last pass, the sentinel's length (4) is added to the sum:

```
    prompt for input; read input ("quit").
    add input length (4) to the sum.
```

- This is a fencepost problem.
  - Must read *N* lines, but only sum the lengths of the first *N*-1.

A fencepost solution

```
sum = 0.
prompt for input; read input.             // place a "post"

while (input is not the sentinel) {
    add input length to the sum.                  // place a "wire"
    prompt for input; read input.                 // place a "post"
}
```

- Sentinel loops often utilize a fencepost "loop-and-a-half" style solution by pulling some code out of the loop.

Correct code

```
Scanner console = new Scanner(System.in);
int sum = 0;

// pull one prompt/read ("post") out of the loop
System.out.print("Type a word (or \"quit\" to exit): ");
String response = console.next();

while (!response.equals("quit")) {
    sum += response.length();    // moved to top of loop
    System.out.print("Type a word (or \"quit\" to exit): ");
    response = console.next();
}

System.out.println("You typed a total of " + sum + " characters.");
```

Sentinel as a constant

```
public static final String SENTINEL = "quit";
...

Scanner console = new Scanner(System.in);
int sum = 0;

// pull one prompt/read ("post") out of the loop
System.out.print("Type a word (or \"" + SENTINEL + "\" to exit): ");
String response = console.next();

while (!response.equals(SENTINEL)) {
    sum += response.length();    // moved to top of loop
    System.out.print("Type a word (or \"" + SENTINEL + "\" to exit):
  ");
    response = console.next();
}

System.out.println("You typed a total of " + sum + " characters.");
```