# Objectives

- Object Oriented Programming
  - OOP review
  - Black-box programming
  - Creating classes in Java
    - State
    - Constructor
    - Methods

# Review

- True or False: you can call methods on an array, e.g.,
  `int[] fibNums = {1, 1, 2, 3, 5};`
- What are some Python → Java Gotchas?
- static
  - What does `static` mean?
  - When should we make a method static?
  - What does a static method have access to?
  - How do you call a static method?

## Review: Object-Oriented Programming

- What is OO programming?
  - What are its components?
- What are its benefits?

## Review: Classes & Objects

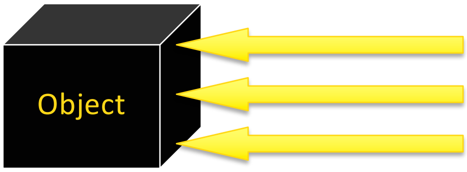
- **Classes** define template from which objects are made
  - “Cookie cutters”
  - Define **state** (aka fields or attributes)
  - Define **behavior**
- Many objects can be created of a class
  - Object: the cookie!
  - Ex: Many Mustangs created from Ford’s “blueprint”
  - Object is an ***instance*** of the class

# Constructors

- **Constructor:** a special method that constructs and initializes an object
  - After construction, can call methods on object

# Black-Box Programming

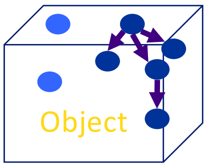
- ***How*** object does something doesn't matter
  - Example: if object *sorts*, does not matter to API user if implements merge or quick sort
- ***What*** object does matters (its **functionality**)
  - What object *exposes* to other objects
  - Referred to as "**black-box programming"** or **encapsulation**

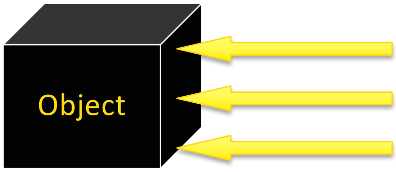


- Has public **interface** that others can use
- Hides state from others

# Discussion

What is the problem with white-box programming?





Others can see and manipulate object's internals

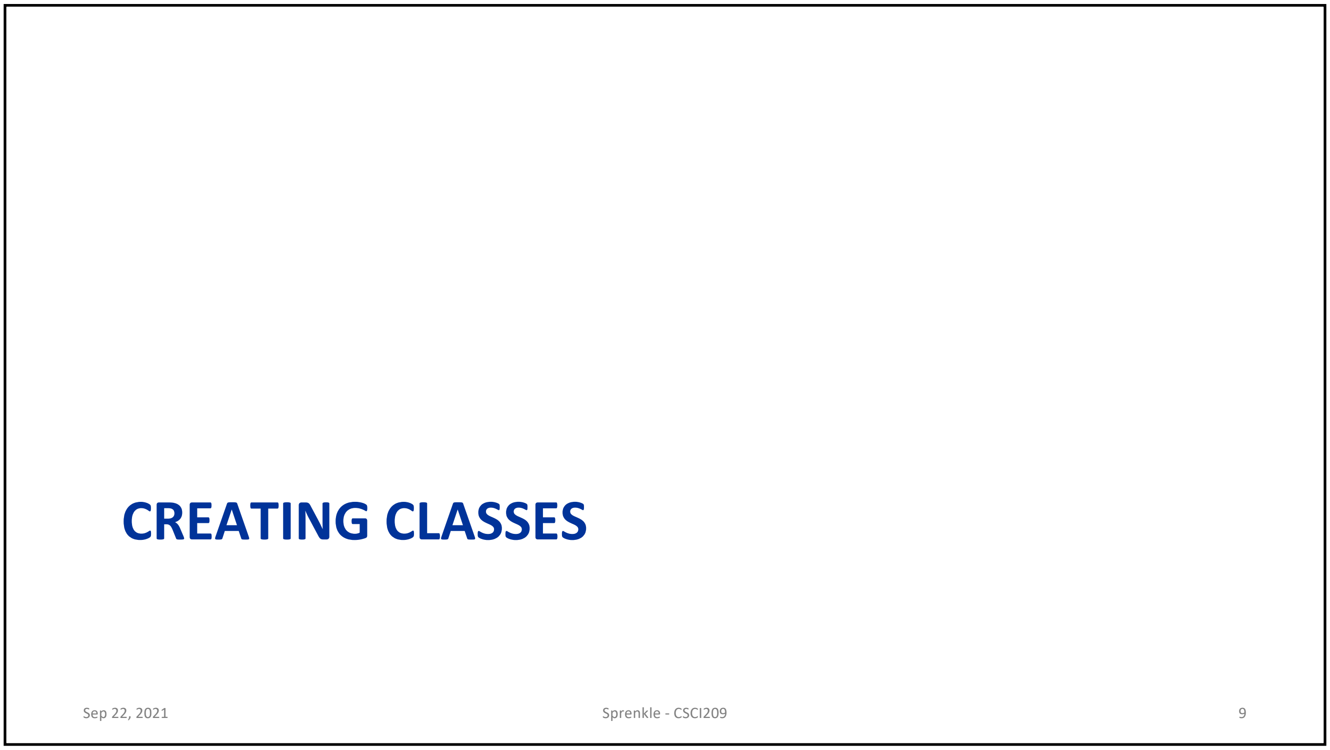
- May have unintended consequences

Java's structure helps us enforce black-box programming

# Access Modifiers

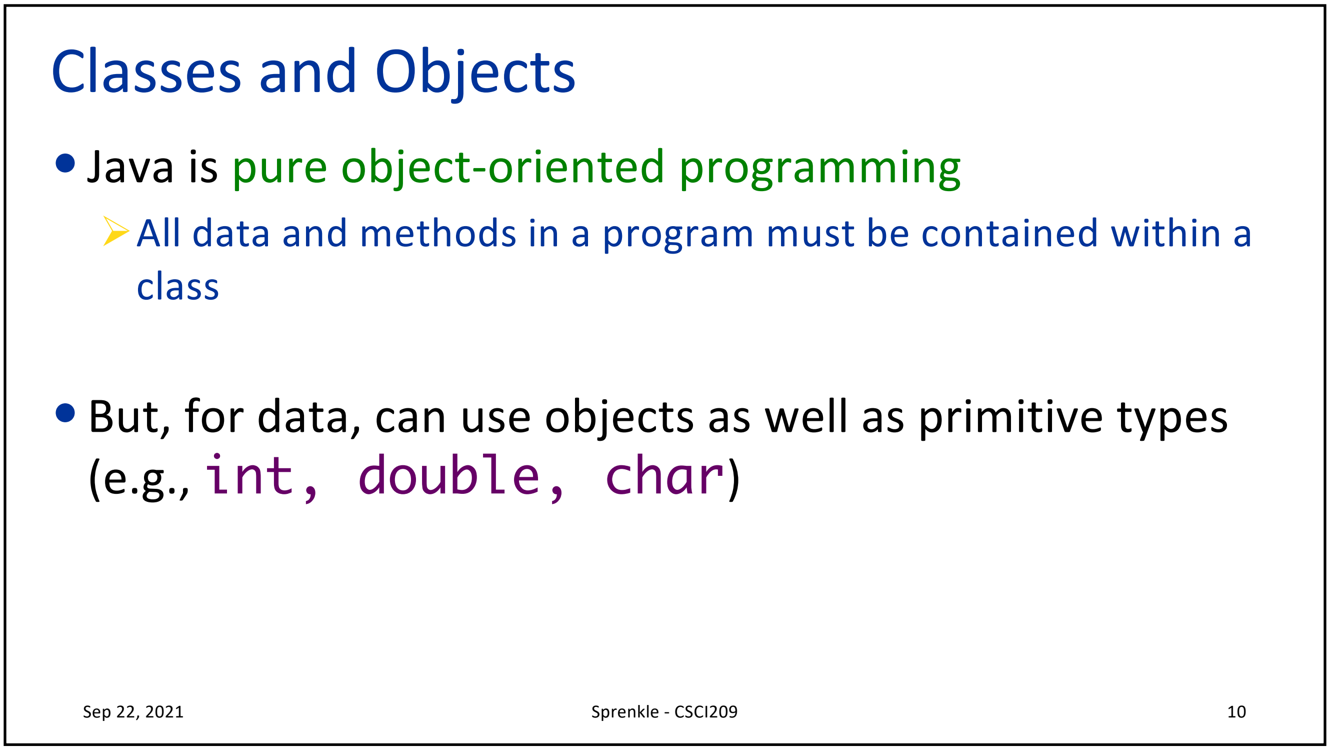
- A `public` method (or instance field) means that any object *of any class* can directly access the method (or field)
  - Least restrictive
- A `private` method (or instance field) means that any object *of the same class* can directly access this method (or field)
  - Most restrictive
- Additional access modifiers will be discussed with inheritance

In general, what access modifiers will we use for methods? For instance fields?



## CREATING CLASSES

9



## Classes and Objects

- Java is pure object-oriented programming
  - All data and methods in a program must be contained within a class
- But, for data, can use objects as well as primitive types (e.g., `int, double, char`)

10

# Example: Chicken class

- State
  - Name, weight, height
- Behavior
  - Accessor methods
    - getWeight, getHeight, getName
    - Convention: “get” for “getter” methods
  - Mutator methods
    - feed: adds weight and height when bird eats
    - setName

# General Java Class Structure

```
public class ClassName {

   // --------- INSTANCE VARIABLES ---------------
   // define variables that represent object’s state
   private int inst_var;

   // --------- CONSTRUCTORS ---------------
   public ClassName() {
      // initialize data structures
   }

   // ----------- METHODS ------------
   public int getInfo() {
      return inst_var;
   }
}
```

Note: instance variables are private and methods are public

# Example: Chicken class

**Discussion**: data types for state variables?

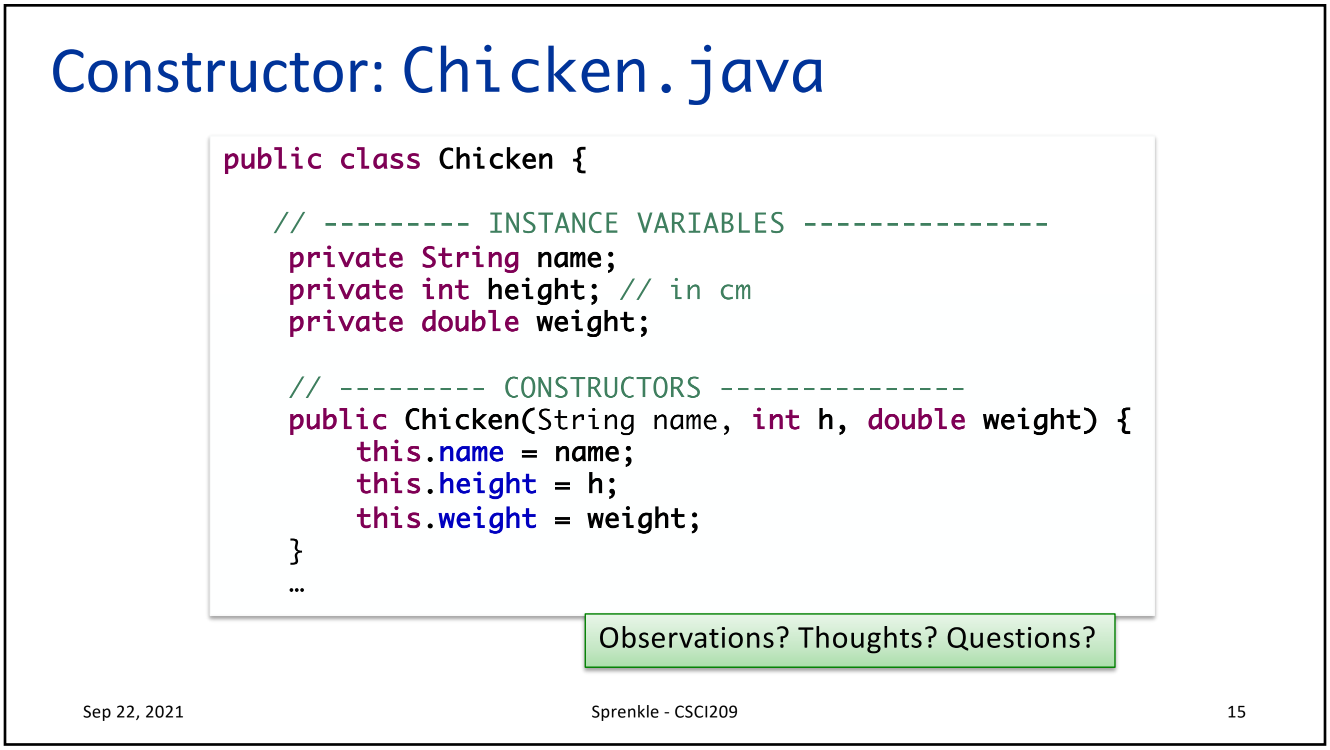
- State
  - ➢ Name, weight, height
- Behavior
  - ➢ Accessor methods
    - `getWeight, getHeight, getName`
    - Convention: “get” for “getter” methods
  - ➢ Mutator methods
    - `feed`: adds weight, height
    - `setName`
      - ➢ Convention: “set” for “setter” methods

# Instance Variables: Chicken.java

```
public class Chicken {

   // --------- INSTANCE VARIABLES ---------------
    private String name;
    private int height; // in cm
    private double weight; // in lbs
```

Instance variables are declared, with access modifier
All instance variables are `private`



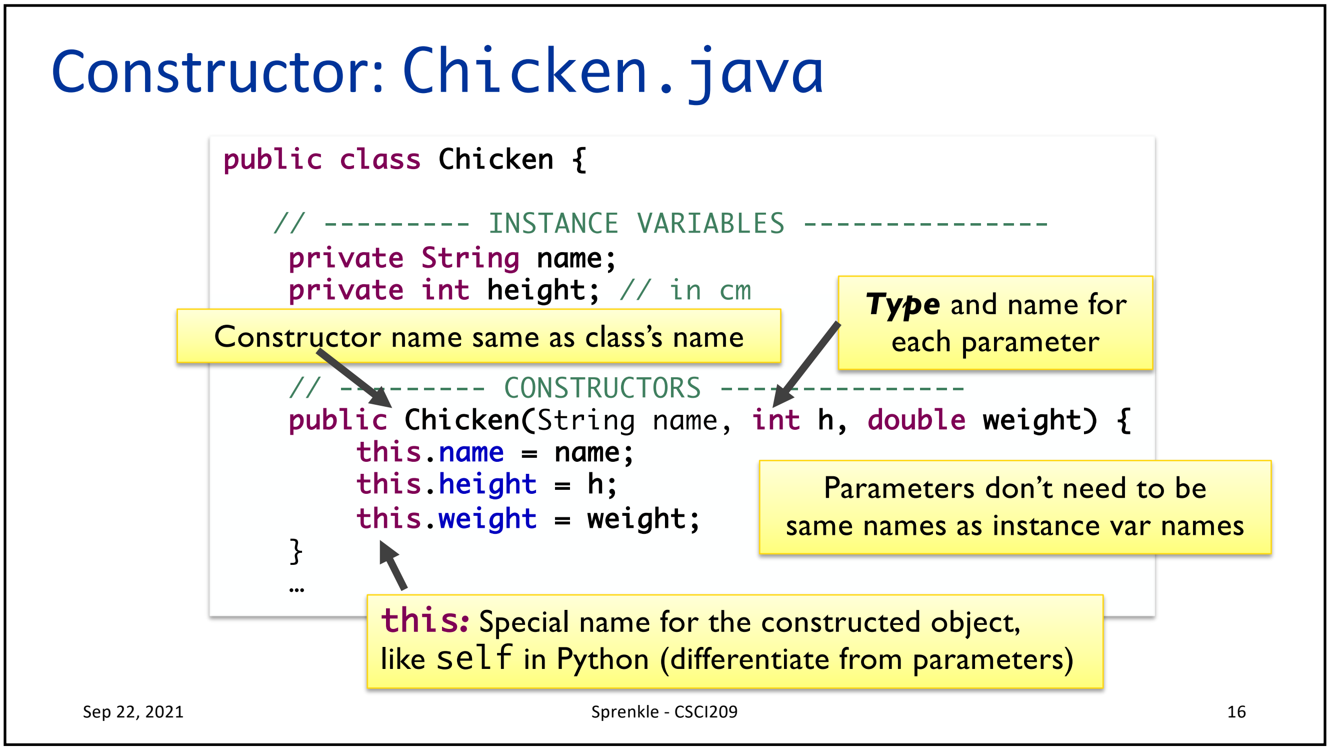
# Constructor: Chicken.java

```
public class Chicken {

   // --------- INSTANCE VARIABLES ---------------
    private String name;
    private int height; // in cm
    private double weight;

    // --------- CONSTRUCTORS ---------------
    public Chicken(String name, int h, double weight) {
        this.name = name;
        this.height = h;
        this.weight = weight;
    }
    …
```

Observations? Thoughts? Questions?



# Constructor: Chicken.java

```
public class Chicken {

   // --------- INSTANCE VARIABLES ---------------
    private String name;
    private int height; // in cm

    // --------- CONSTRUCTORS ---------------
    public Chicken(String name, int h, double weight) {
        this.name = name;
        this.height = h;
        this.weight = weight;
    }
    …
```

Constructor name same as class's name

***Type*** and name for each parameter

Parameters don't need to be same names as instance var names

**this:** Special name for the constructed object, like `self` in Python (differentiate from parameters)

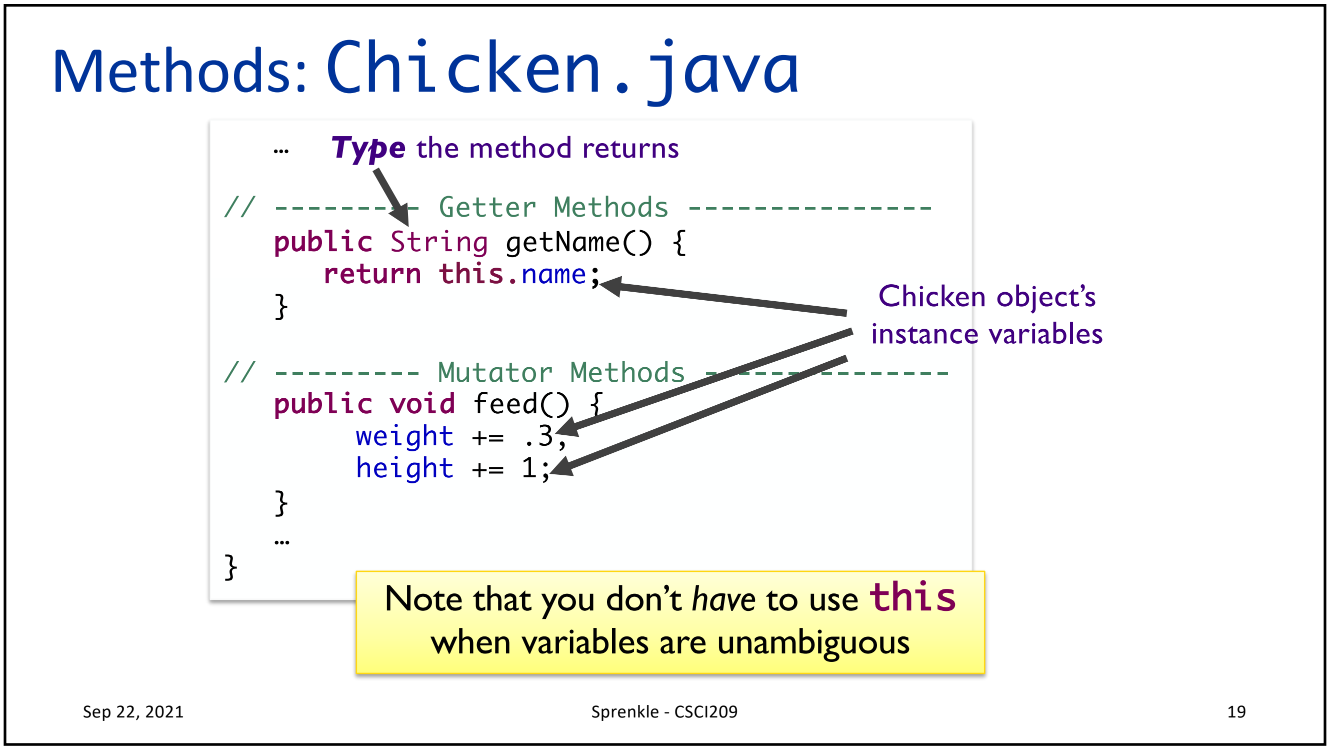
# Constructors

- **Constructor:** a special method that constructs and initializes an object
  - After construction, can call methods on object
- A constructor has the same name as its class
- Like `__init__` in Python

# Example: `Chicken` class

- State
  - Name, weight, height
- Behavior
  - Accessor methods
    - `getWeight, getHeight, getName`
    - Convention: “get” for “getter” methods
  - Mutator methods
    - `feed`: adds weight, height to this Chicken
    - `setName`

***Discussion***: What are the methods’ ***input*** (parameters) and ***output*** (what is returned)?



# Methods: Chicken.java

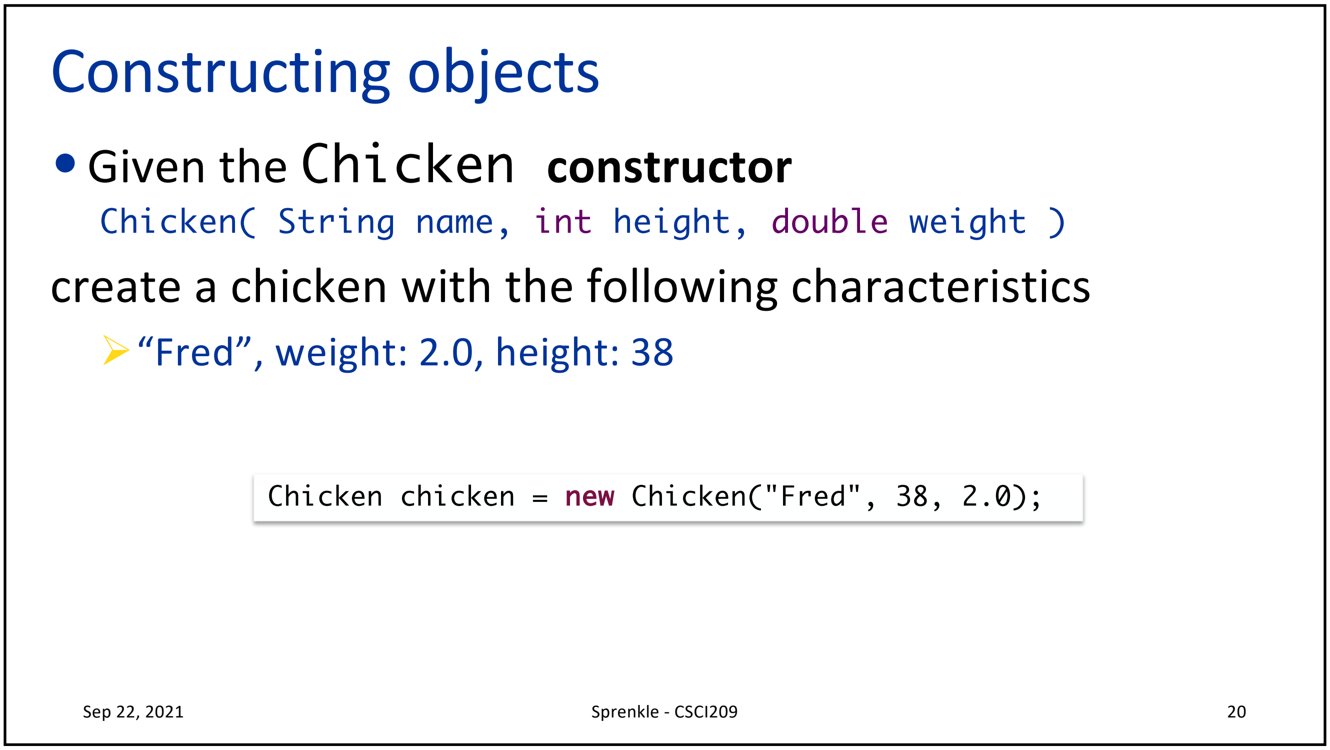
*Type* the method returns

```
    …
// --------- Getter Methods ----------------
   public String getName() {
      return this.name;
   }

// --------- Mutator Methods ---------------
   public void feed() {
        weight += .3;
        height += 1;
   }
   …
}
```

Chicken object's instance variables

Note that you don't *have* to use `this` when variables are unambiguous



# Constructing objects

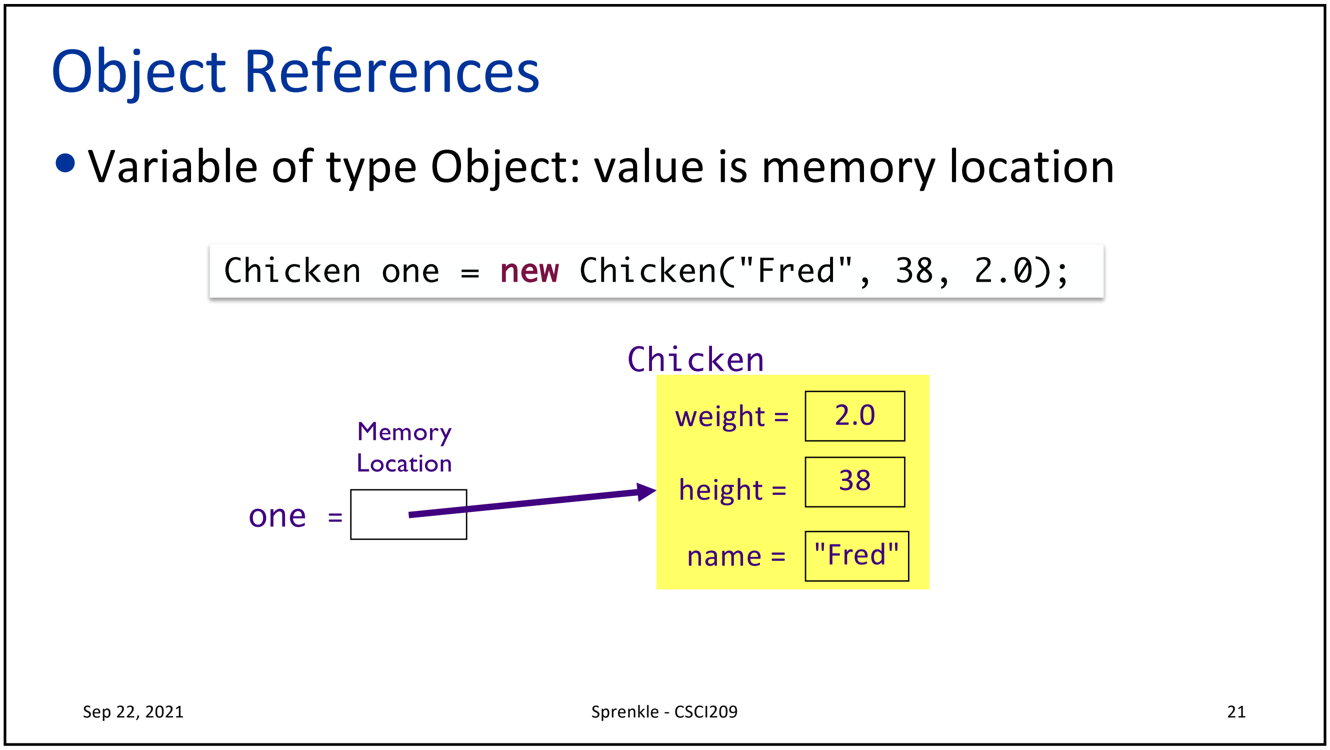
- Given the `Chicken` **constructor**

  `Chicken( String name, int height, double weight )`

create a chicken with the following characteristics

  - "Fred", weight: 2.0, height: 38

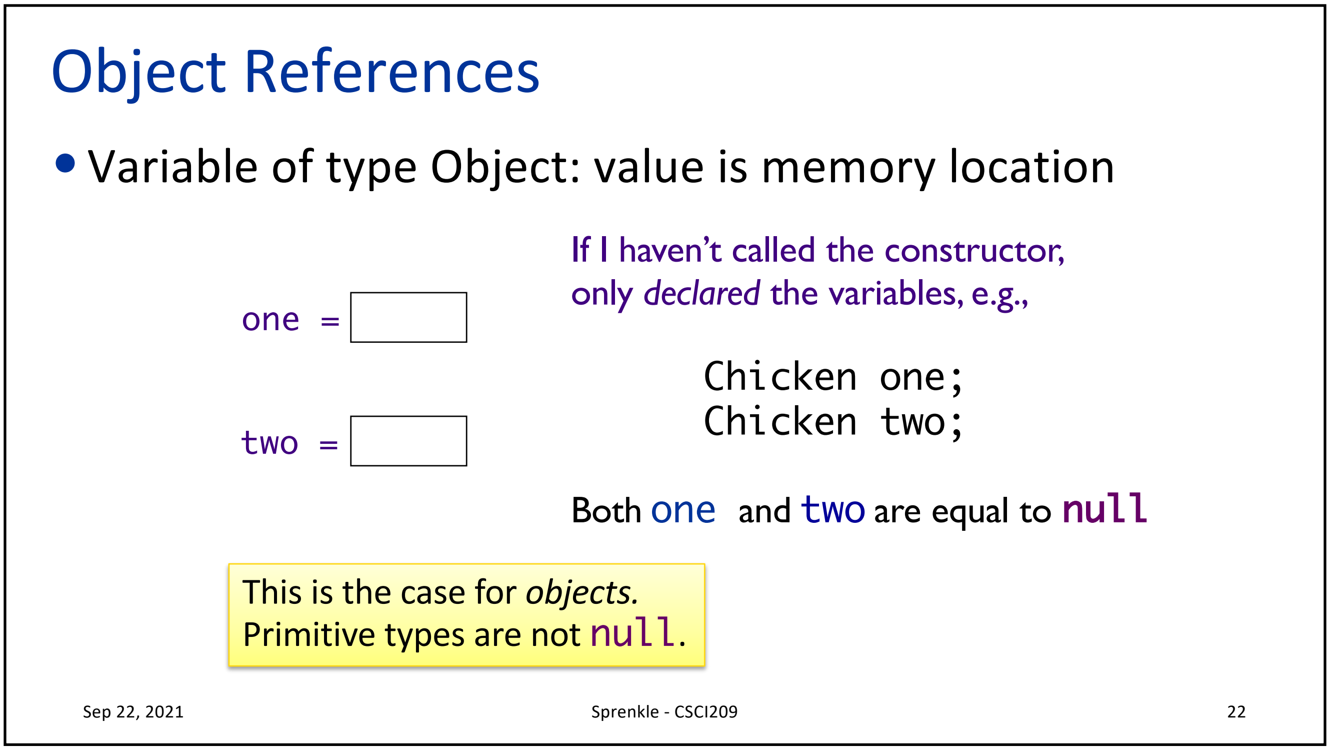
```
Chicken chicken = new Chicken("Fred", 38, 2.0);
```



# Object References

- Variable of type Object: value is memory location

```
Chicken one = new Chicken("Fred", 38, 2.0);
```



# Object References

- Variable of type Object: value is memory location

If I haven't called the constructor, only *declared* the variables, e.g.,

```
Chicken one;
Chicken two;
```

Both one and two are equal to null

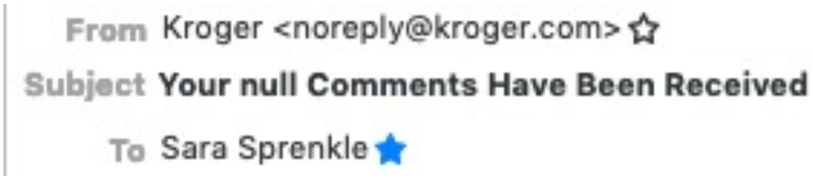
This is the case for *objects.*
Primitive types are not null.

# Null Object Variables

- An object variable can be explicitly set to `null`
  - Means that the object variable does not currently refer to any object
- Can test if an object variable is set to `null`

```
Chicken chick = null;
   … … …
if (chick == null) {
       . . .
}
```

# Recall This Error Message



From Kroger <noreply@kroger.com>

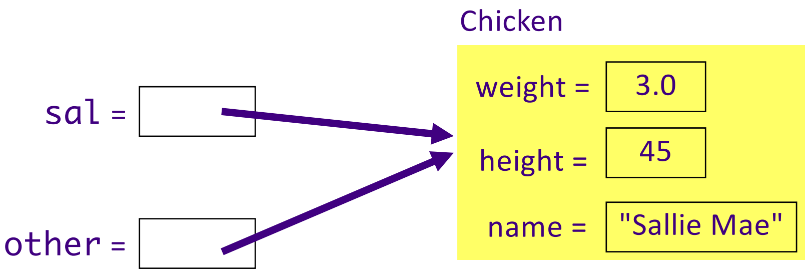
Subject **Your null Comments Have Been Received**

To Sara Sprenkle

# Multiple Object Variables

- More than one object variable can refer to the same object

```
Chicken sal = new Chicken("Sallie Mae");
Chicken other = sal;
```



# Chicken's `main` method

- Where we'll do testing
  1. Create object
  2. Call methods
  3. Verify methods' results are what you expect

- When done testing, can move tests into separate test method
- Later: better ways to test

# Class Development Process

1. Determine state
   - ➢ Declare state at top of class
2. Define constructor
   - ➢ Call constructor/create an object
3. Repeat
   - ➢ Write method or constructor
   - ➢ Test new method or constructor

# MORE ON OBJECT INITIALIZATION

## Default Object State Initialization

- If instance field is not explicitly set in constructor, automatically set to default value
  - Numbers set to zero
  - Booleans set to `false`
  - Object variables set to `null`
  - *Local variables are not assigned defaults*
- **Do not** rely on defaults
  - Code is harder to understand

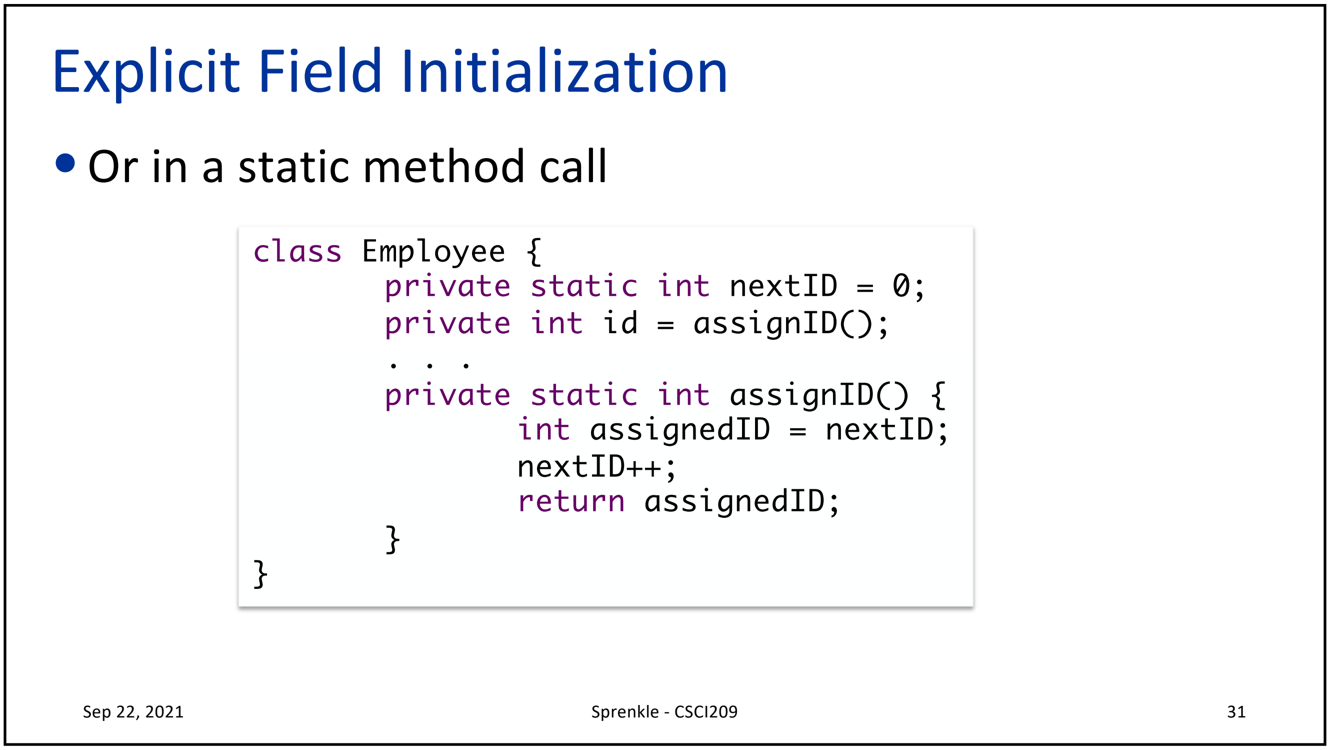
***Clean Code Recommendation:***
Set all instance fields in the constructor(s)

## Explicit Field Initialization

- If more than one constructor needs an instance field set to same value, the field can be set explicitly in the field declaration

```
class Chicken {
        private String name = "";
        . . .
}
```

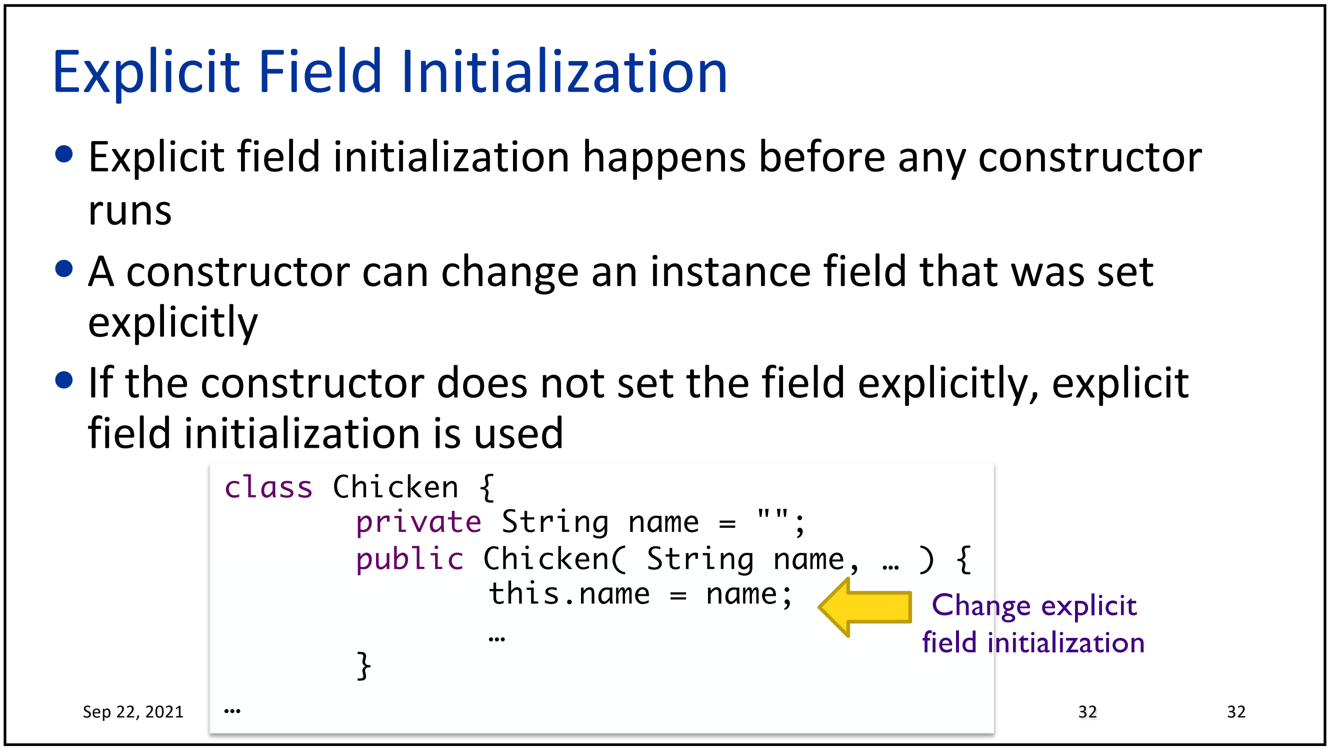
Set value here for
all constructors



## Explicit Field Initialization

- Or in a static method call

```
class Employee {
        private static int nextID = 0;
        private int id = assignID();
        . . .
        private static int assignID() {
                int assignedID = nextID;
                nextID++;
                return assignedID;
        }
}
```



## Explicit Field Initialization

- Explicit field initialization happens before any constructor runs
- A constructor can change an instance field that was set explicitly
- If the constructor does not set the field explicitly, explicit field initialization is used

```
class Chicken {
        private String name = "";
        public Chicken( String name, … ) {
                this.name = name;
                …
        }
…
```

Change explicit field initialization

# final keyword

- An instance field can be final
- final instance fields **must** be set in the constructor or in the field declaration
  - Cannot be changed ***after object is constructed***

```
private final String dbName = "invoices";
private final String id;
…
public MyObject( String id ) {
        this.id = id;
}
```

# TODO

- Assignment 4 – due Tuesday at 11:59 p.m.
  - OO programming
  - Recommendation
    - Do parts 1 and 2 before Friday’s class
    - Do part 3 before Monday’s class
- Textbook – Read “Defining Classes in Java” up to but not including Inheritance