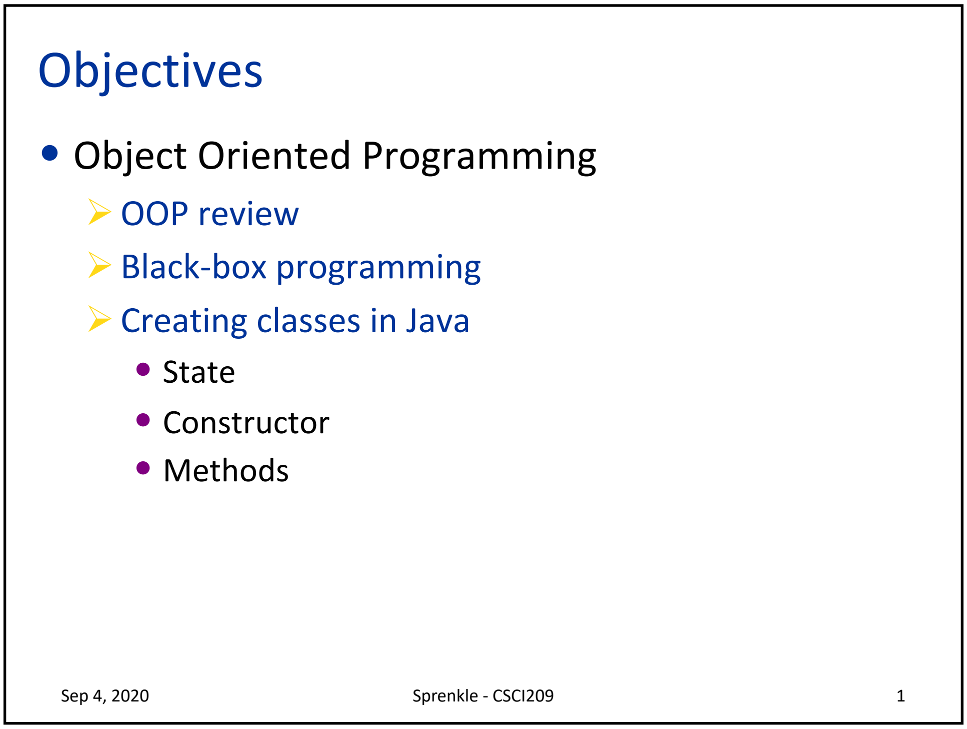
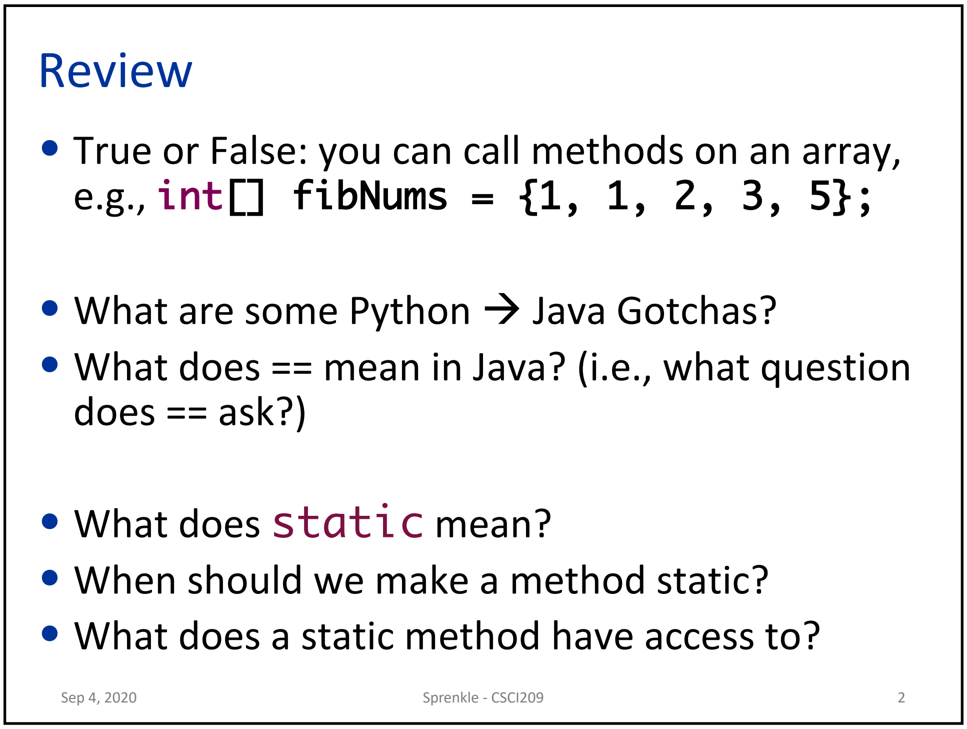


Objectives

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

1



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?
- What does == mean in Java? (i.e., what question does == ask?)
- What does `static` mean?
- When should we make a method static?
- What does a static method have access to?

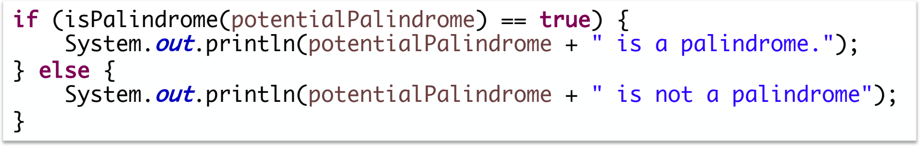
2

Assignment 3 Feedback: Remember good programming practices

- No side effects to methods
  - If method does not say that it is printing something, don’t print something
    - Printing while debugging is fine; remove print statements before final submission
  - Want method to reusable → print statements may not be what others want
- Leverage existing APIs
  - `StringBuilder` has `replace` and `reverse` methods
    - Likely more efficient than anything you write

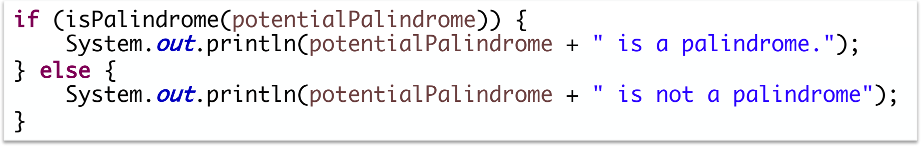
Assignment 3 Discussion

Move from this (when you were new to programming):

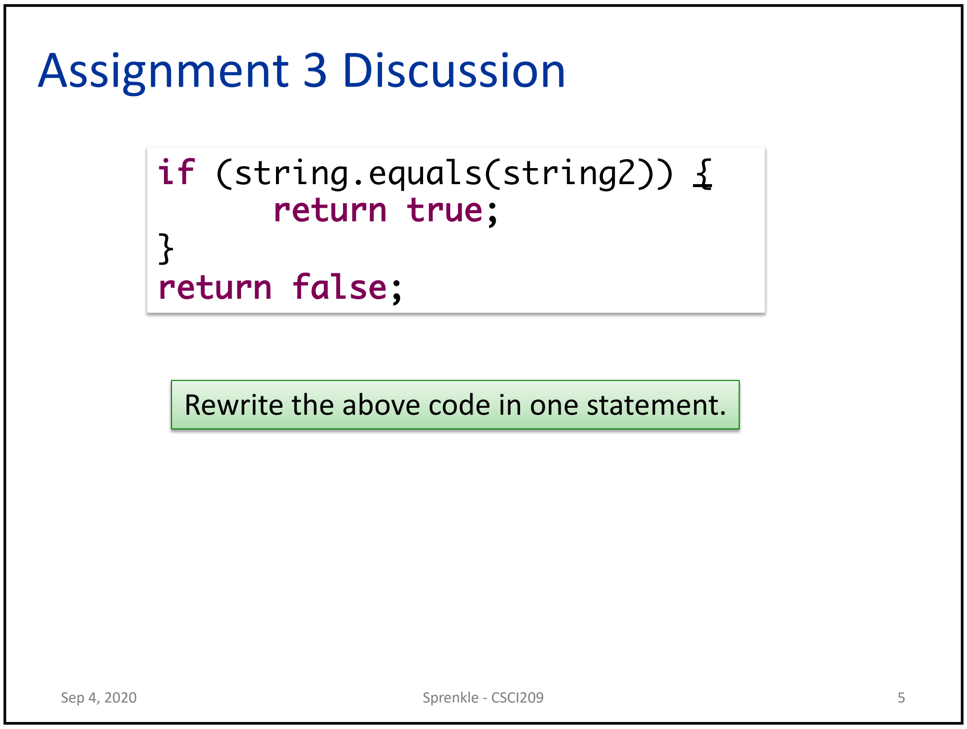


```
if (isPalindrome(potentialPalindrome) == true) {
    System.out.println(potentialPalindrome + " is a palindrome.");
} else {
    System.out.println(potentialPalindrome + " is not a palindrome");
}
```

To this (at least your 3rd programming course):



```
if (isPalindrome(potentialPalindrome)) {
    System.out.println(potentialPalindrome + " is a palindrome.");
} else {
    System.out.println(potentialPalindrome + " is not a palindrome");
}
```

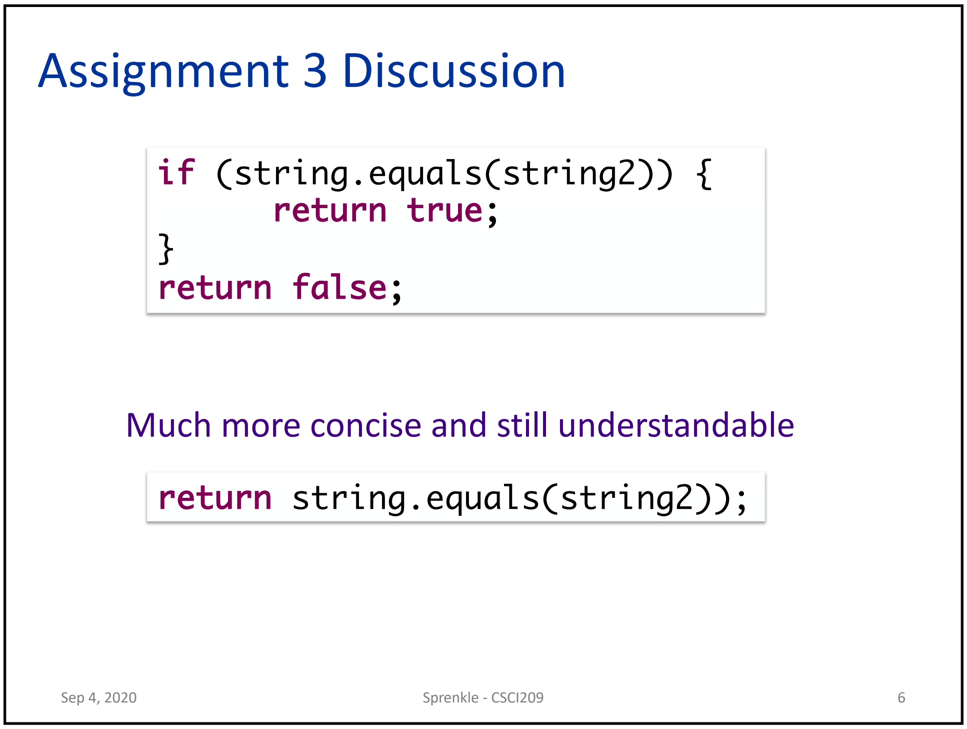


Assignment 3 Discussion

```java
if (string.equals(string2)) {
        return true;
}
return false;
```

Rewrite the above code in one statement.

5



Assignment 3 Discussion

```java
if (string.equals(string2)) {
        return true;
}
return false;
```

Much more concise and still understandable

```java
return string.equals(string2);
```

6

Review: Object-Oriented Programming

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

Review: Classes & Objects

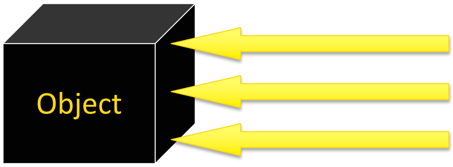
- **Classes** define template from which objects are made
  - "Cookie cutters"
  - Define **state** (aka fields or attributes)
  - Define **behavior**
- Many objects can be created of a class
  - Object: the cookie!
  - Ex: Many Mustangs created from Ford's "blueprint"
  - Object is an ***instance***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

Constructors

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

Black-Box Programming

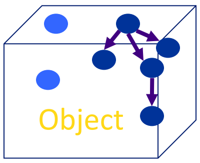
- ***How*** object does something doesn't matter
  - Example: if object *sorts*, does not matter if uses merge or quick sort
- ***What*** object does matters (its **functionality**)
  - What object *exposes* to other objects
  - Referred to as "**black-box programming"** or **encapsulation**



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

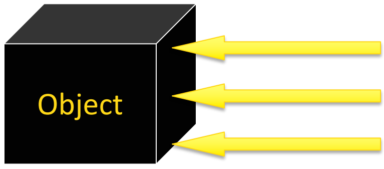
Discussion

What is the problem with white-box programming?



Others can see and manipulate object's internals

- May have unintended consequences

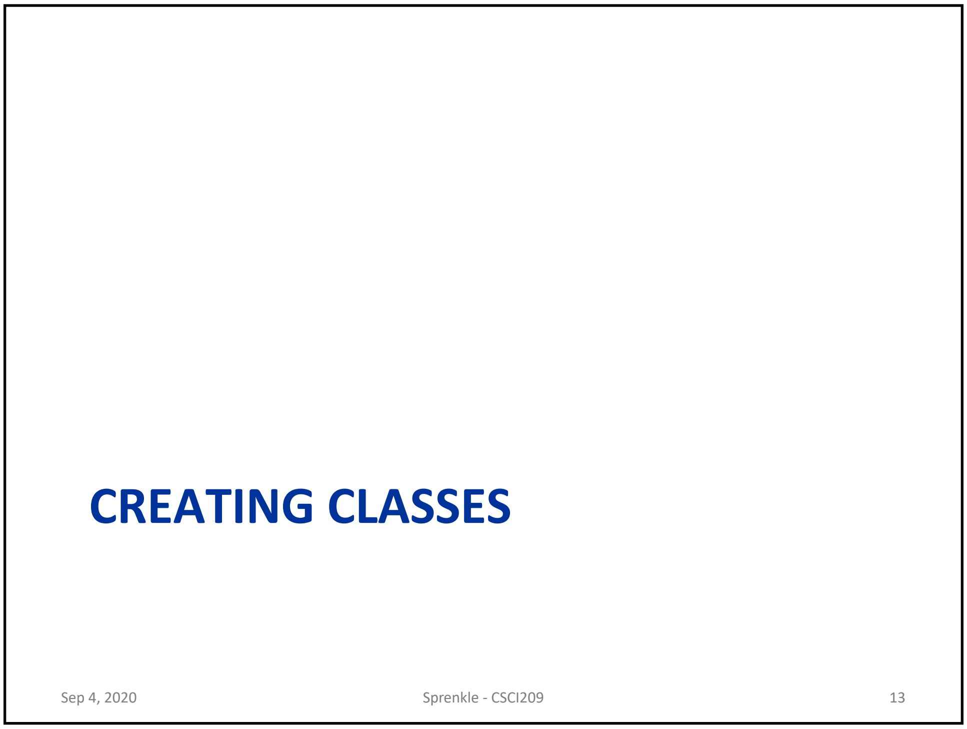


Java's structure helps us enforce black-box programming

Access Modifiers

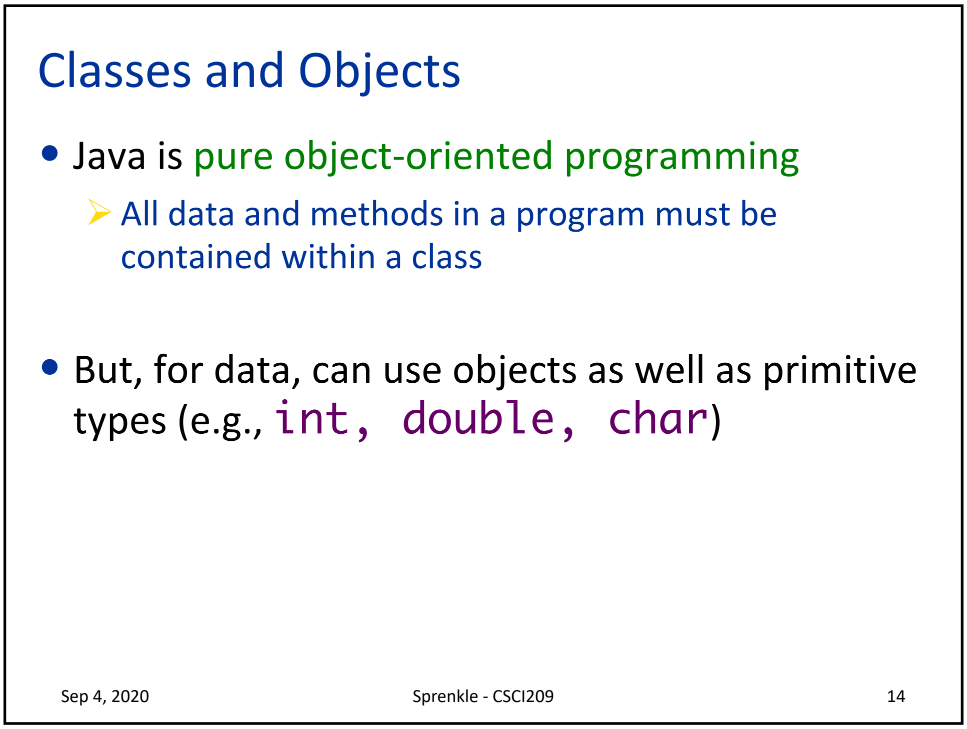
- A **public** method (or instance field) means that any object *of any class* can directly access the method (or field)
  - Least restrictive
- A **private** method (or instance field) means that any object *of the same class* can directly access this method (or field)
  - Most restrictive
- Additional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

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



CREATING CLASSES

13



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`)

14

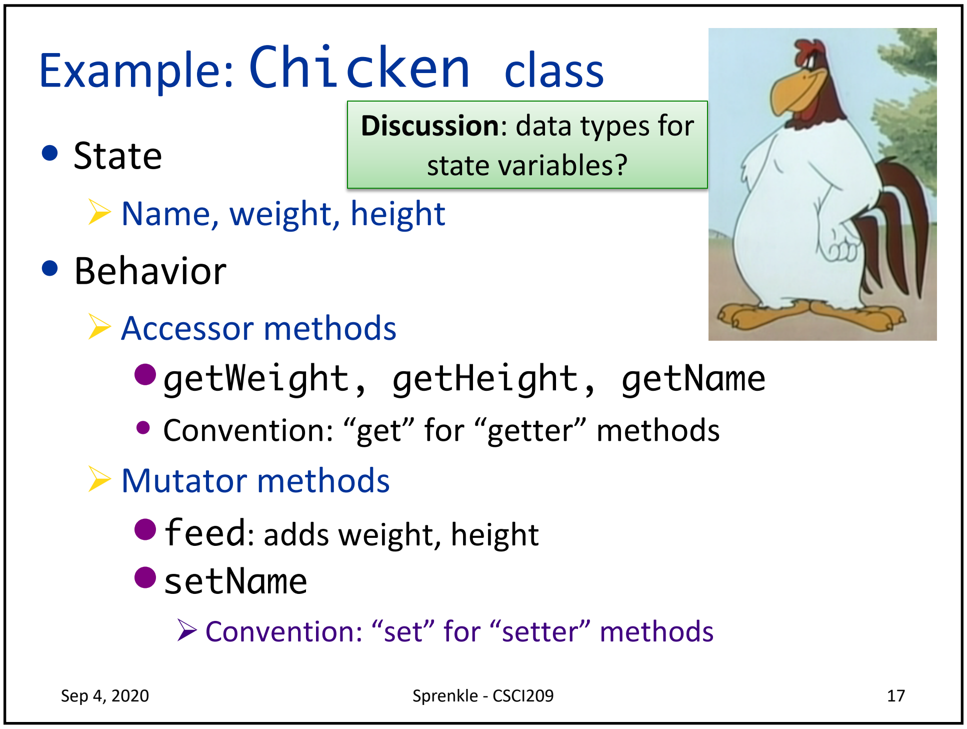
Example: `Chicken` class

- 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

```java
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;
    }
}
```

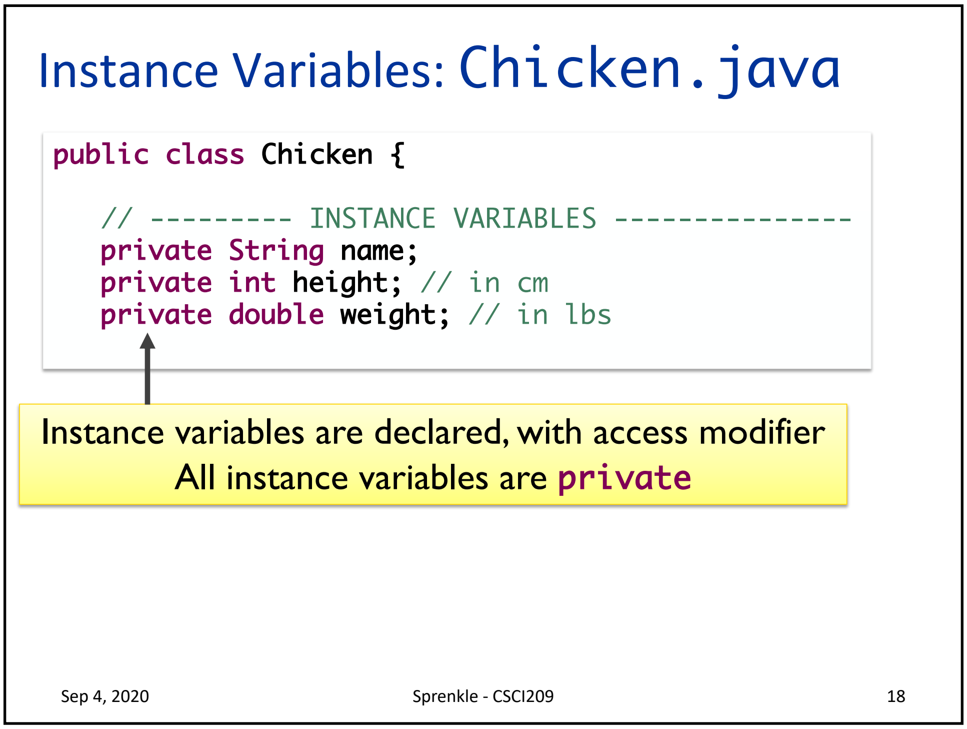
Note: instance variables are `private` and methods are `public`



Example: Chicken class

Discussion: data types for state variables?

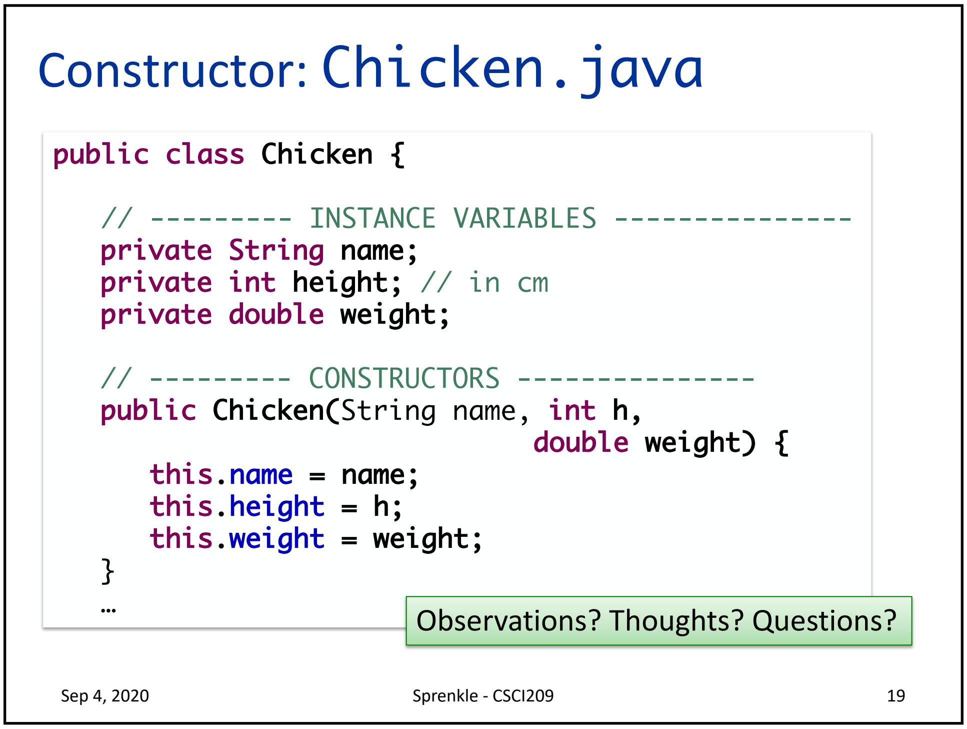
- State
  - Name, weight, height
- Behavior
  - Accessor methods
    - getWeight, getHeight, getName
    - Convention: “get” for “getter” methods
  - Mutator methods
    - feed: adds weight, height
    - setName
      - Convention: “set” for “setter” methods



Instance Variables: Chicken.java

```
public class Chicken {

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

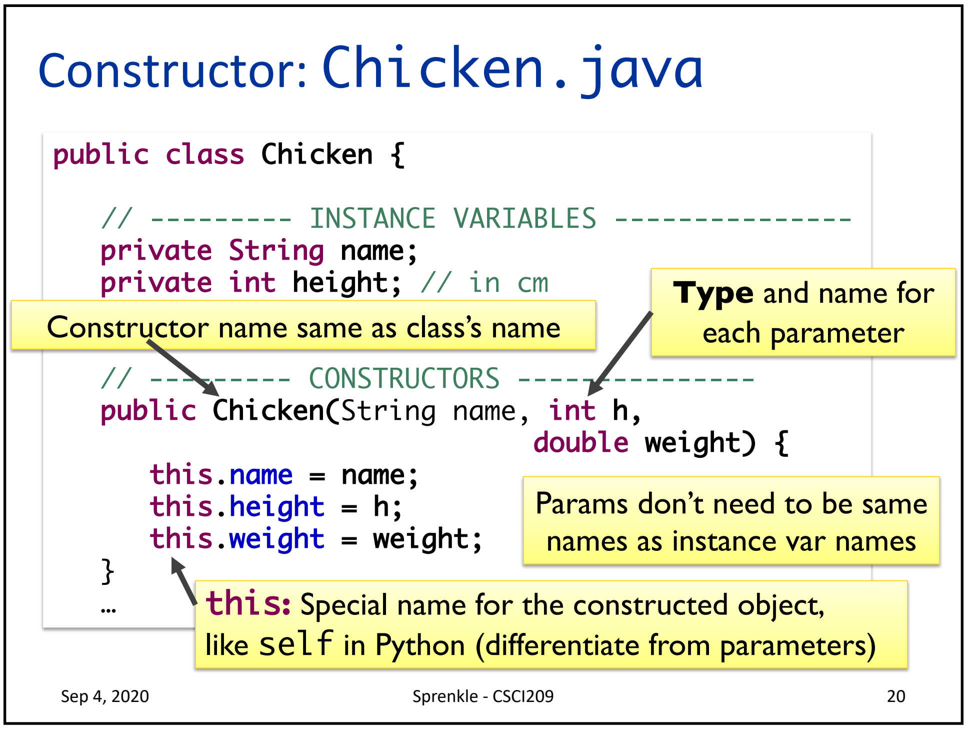
Instance variables are declared, with access modifier
All instance variables are private



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

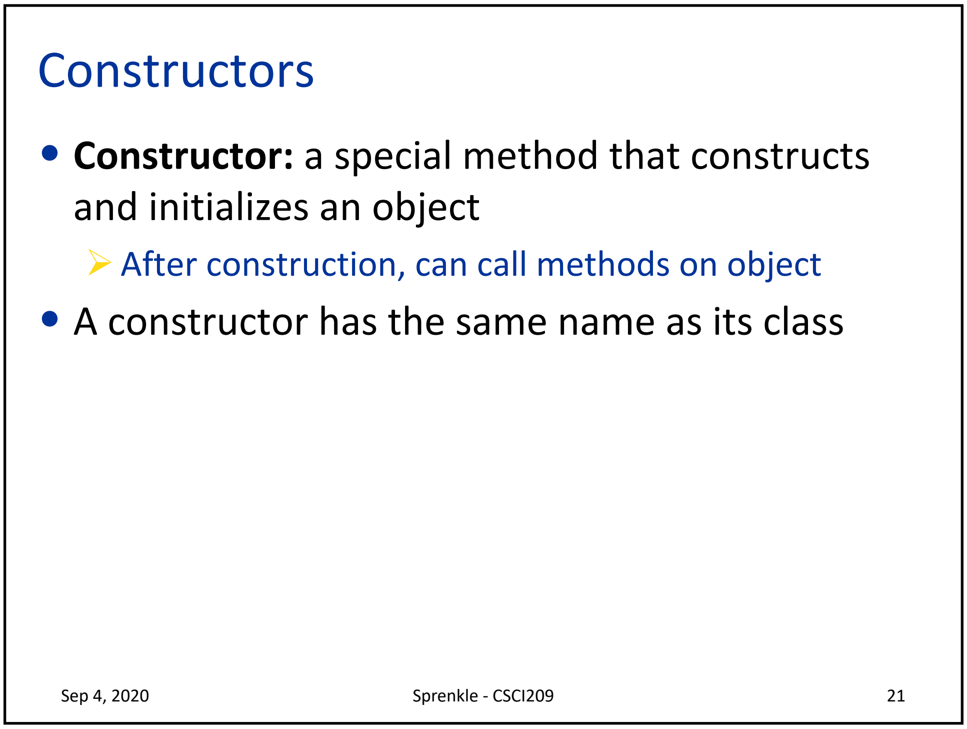
```
public class Chicken {

   // --------- 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

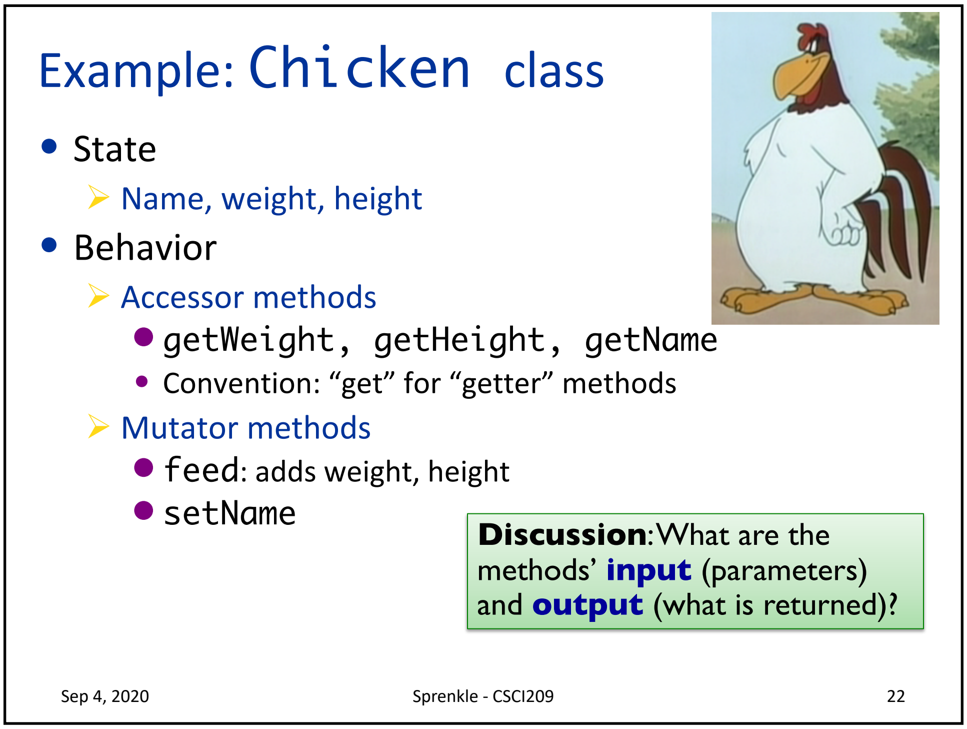
Params don't need to be same names as instance var names

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



Constructors

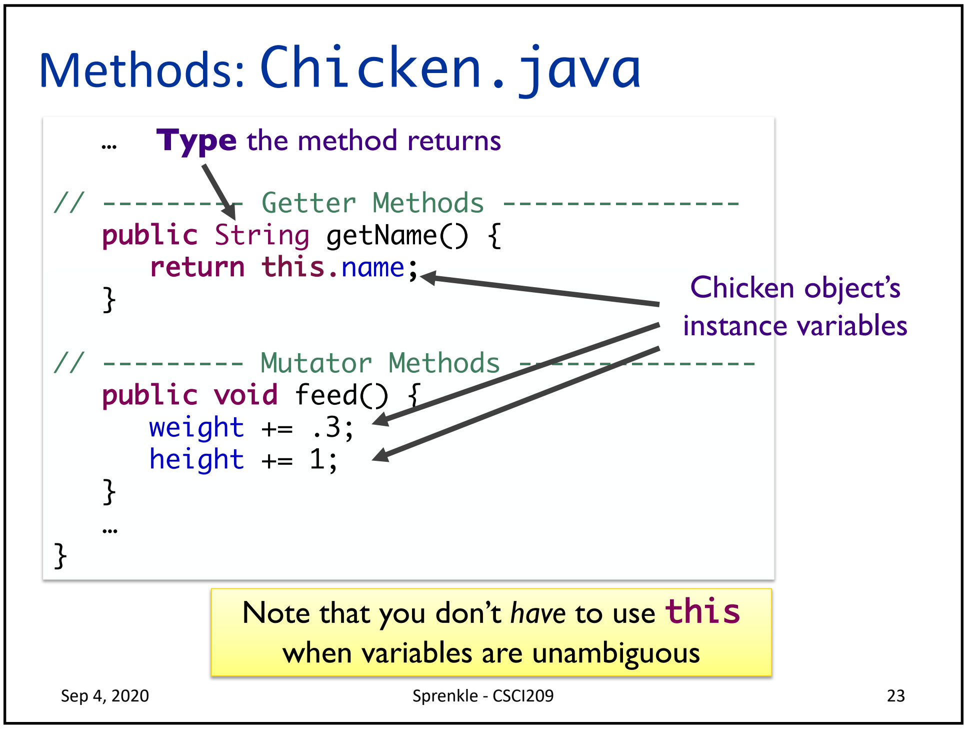
- **Constructor:** a special method that constructs and initializes an object
  - After construction, can call methods on object
- A constructor has the same name as its class



Example: `Chicken` class

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

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



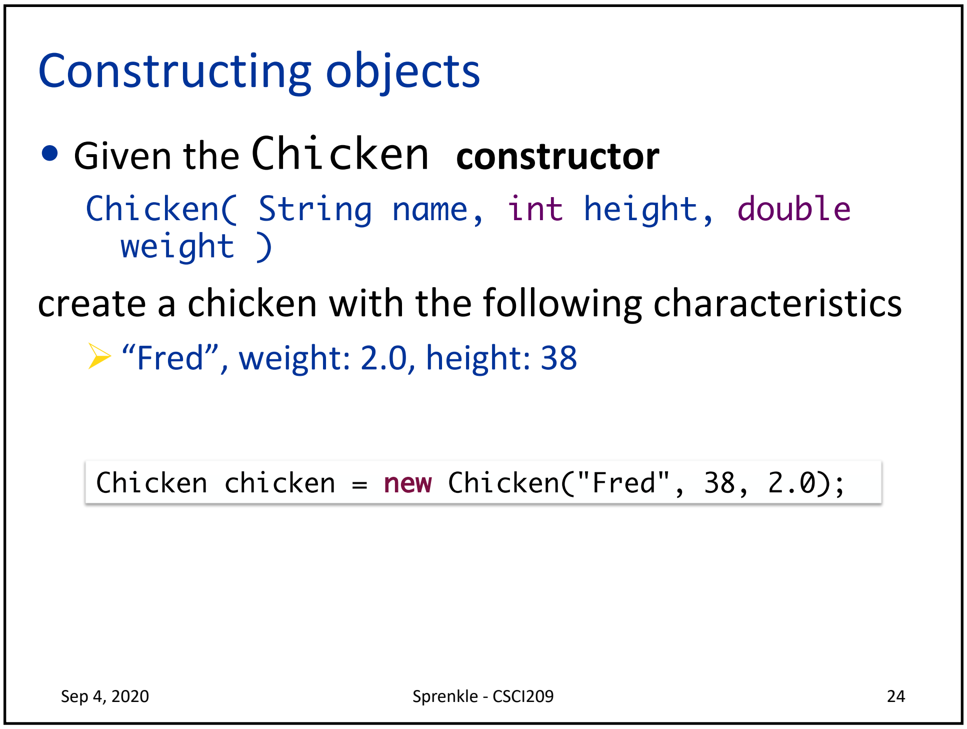
Methods: Chicken.java

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

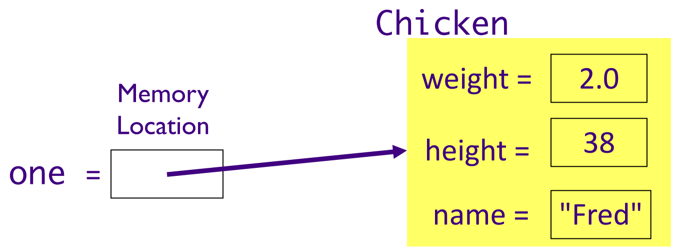
- "Fred", weight: 2.0, height: 38

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

Object References

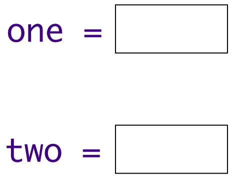
- Variable of type Object: value is memory location

```
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

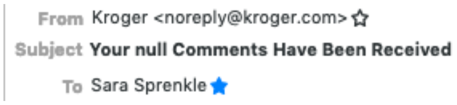
This is the case for *objects*.
Primitive types are not null.

Null Object Variables

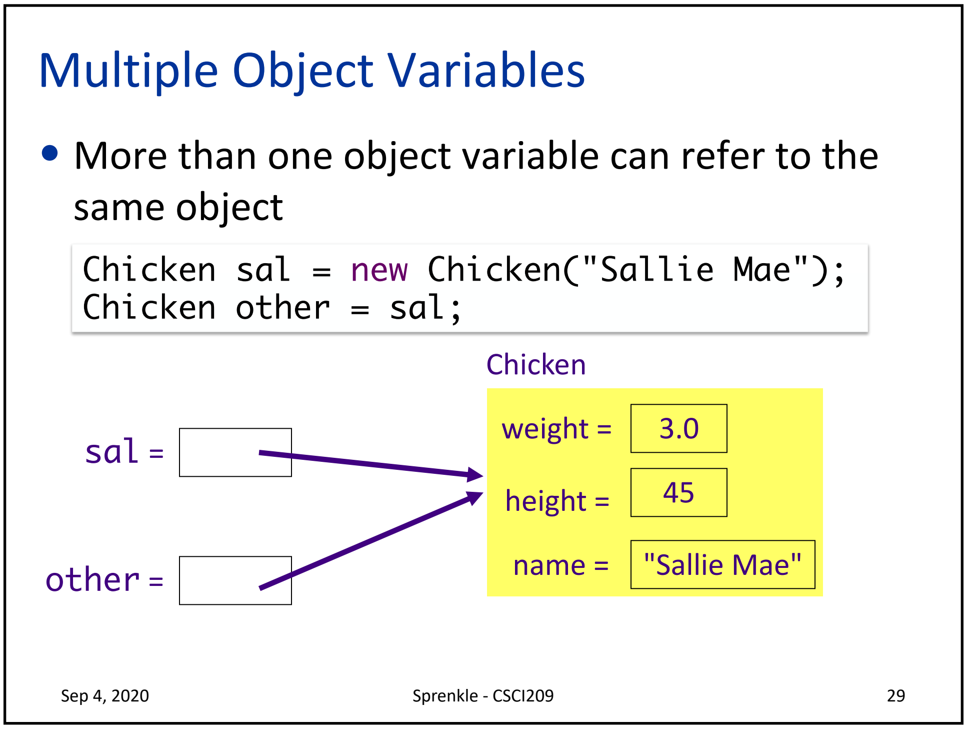
- An object variable can be explicitly set to `null`
  - Means that the object variable does not currently refer to any object
- Can test if an object variable is set to `null`

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

Recall This Error Message



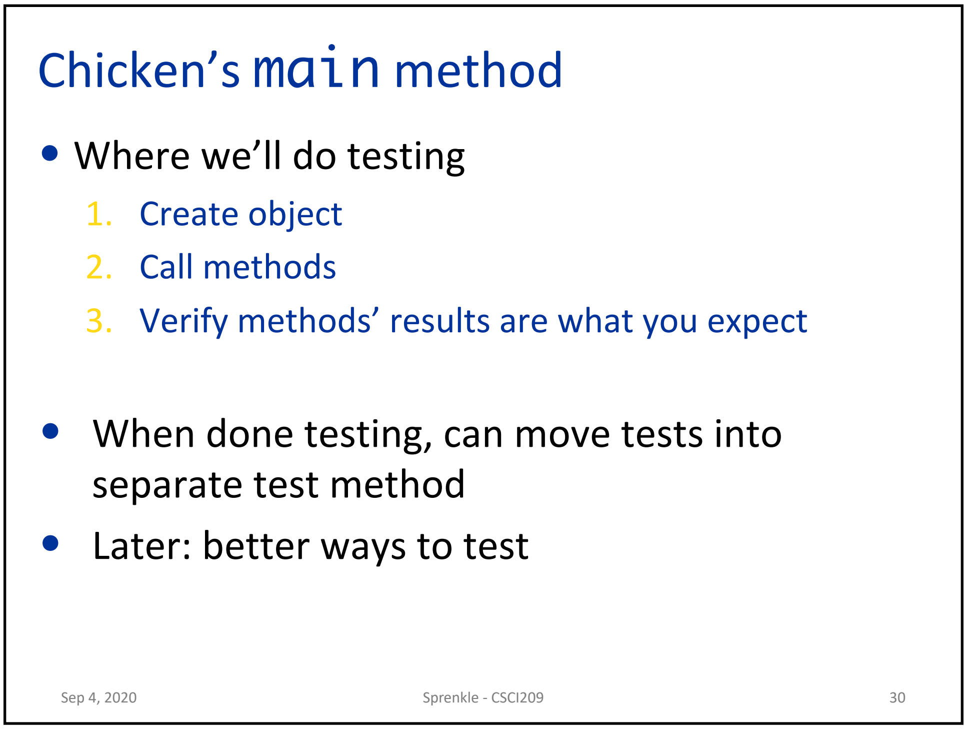
From Kroger <noreply@kroger.com> ☆
Subject **Your null Comments Have Been Received**
To Sara Sprenkle ★



Multiple Object Variables

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

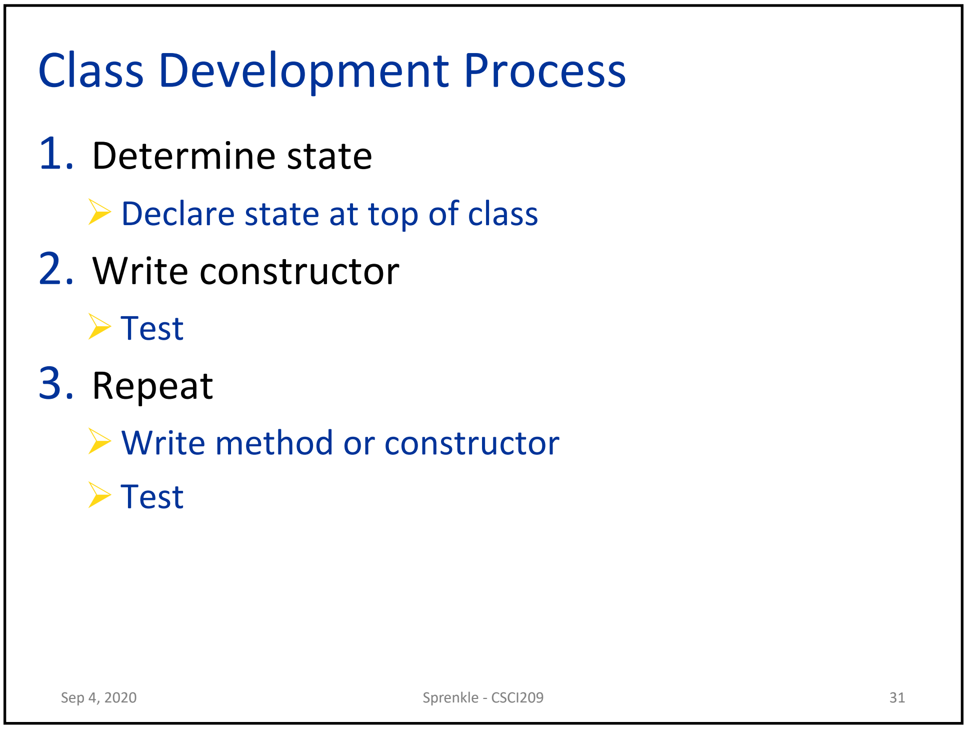
```
Chicken sal = new Chicken("Sallie Mae");
Chicken other = sal;
```



Chicken’s `main` method

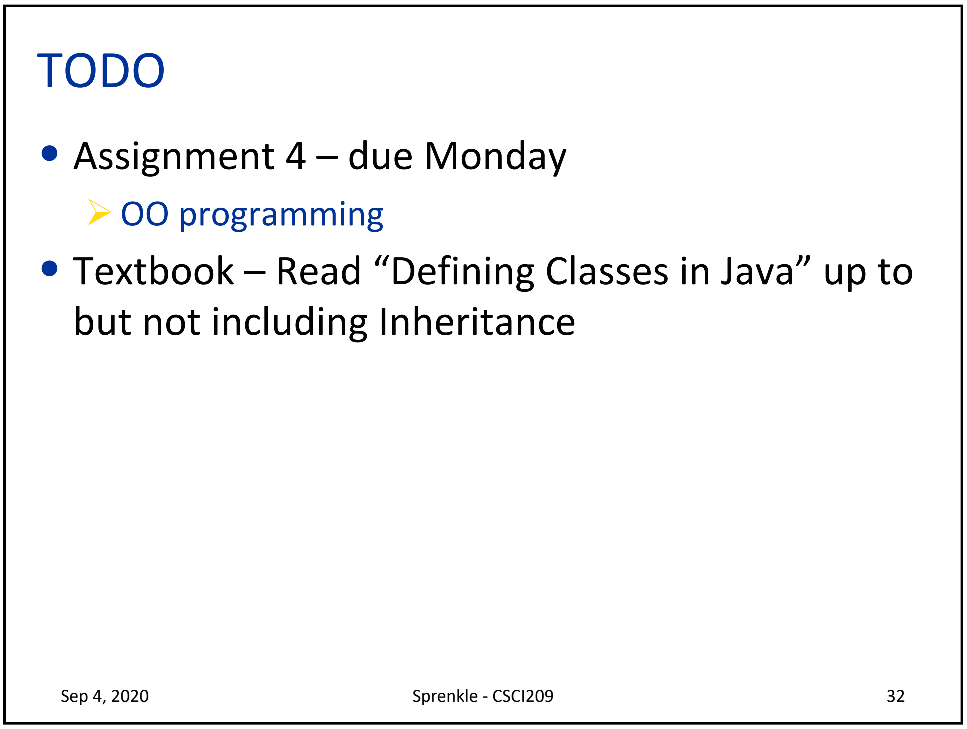
- Where we’ll do testing
  1. Create object
  2. Call methods
  3. Verify methods’ results are what you expect

- 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. Write constructor
   - ➢ Test
3. Repeat
   - ➢ Write method or constructor
   - ➢ Test



TODO

- Assignment 4 – due Monday
  - ➢ OO programming
- Textbook – Read “Defining Classes in Java” up to but not including Inheritance