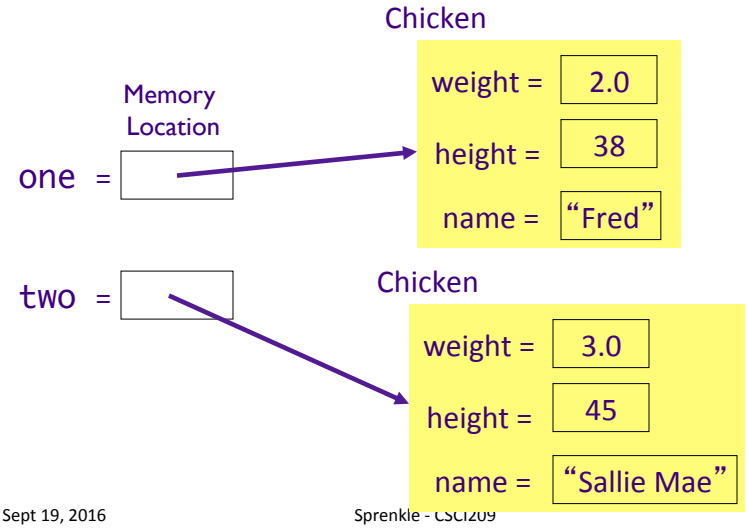
Objectives

- Object-oriented programming in Java
  - Object references
  - Static methods, fields
  - Constructors
  - Default constructors

Object References

The following 3 slides were already in Friday's pre-reading.

- Variable of type object: value is memory location



Object References

- Variable of type object: value is memory location

one =

two =

If I haven't called the constructor, only *declared* the variables:

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

Both one and two are equal to null

Null Object Variables

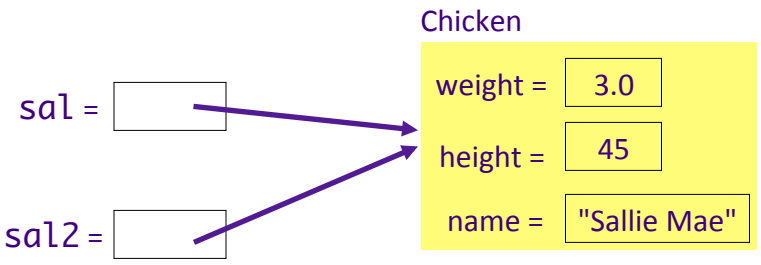
- An object variable can be explicitly set to null
  - Means that the object variable does not currently refer to any object
- Can test if an object variable is set to null

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

Multiple Object Variables

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

```
Chicken sal = new Chicken("Sallie Mae");
Chicken sal2 = sal;
```



STATIC METHODS AND FIELDS

Static Methods/Fields

- For related functionality/data that isn't specific to any particular object

- `java.lang.Math`
  - No constructor (what does that mean?)
  - Static fields: PI, E
  - Static methods:
    - `static double sin(double a)`

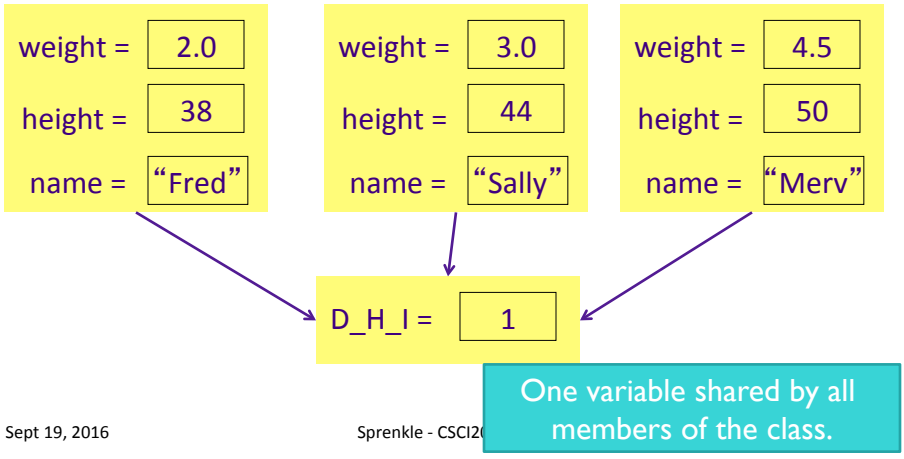
Static Fields

- A static field is used when only one such field **per class** (not object!)
- All objects of a class share **one copy** of the static field

Chicken static field example

```
static int DEFAULT_HEIGHT_INCREASE = 1;
```

A bunch of Chicken objects



Review: Constant Static Fields

- We used a static field to designate a ***class constant***:

```
public class Converter {
      public static final double CM2IN = 2.54;
```

- The `Math` class has a static constant, `PI`
  - Value can be accessed using the `Math` class: `area = Math.PI * r * r;`
  - Do not need an object of the `Math` class to use this constant

Static Fields Example

```
public class Student {
      private static int nextID = 1;
      private int id;
      . . .
}
```

- Each `Student` object has an `id` field, but there is only one `nextID` field, shared among all instances of the class
  - `nextID` field exists even when no `Student` objects have been constructed

How could we use the `nextID` field to create unique IDs?

Static Field Example

```
public class Student {
      private static int nextID = 1;
      private int id = assignID();

      private static int assignID() {
            int r = nextID;
            nextID++;
            return r;
      }
…
```

Static Methods

- Do **not** operate on objects
  - Can**not** access instance fields of their class
- Can access *static fields* of their class
- Similar to Python *functions* that are associated with the class

Why is `main` static?

MORE ON CONSTRUCTORS

More on Constructors

- A class can have **more than one** constructor
  - Whoa! Let that sink in for a bit
- A constructor can have zero, one, or multiple parameters
- A constructor has no return value
- A constructor is always called with the new operator

Constructor Overloading

- Allowing > 1 constructor (or any method) with the same name is called ***overloading***
  - Constraint: Each of the methods that have the same name must have different parameters so that compiler can distinguish between them
    - "different" → *Number* and/or *type*
- Compiler handles ***overload resolution***
  - Process of matching a method call to the correct method by matching the parameters
- No function overloading in Python

Why isn't overloading possible in Python?

Default Constructor

- ***Default constructor*****:** constructor with no parameters
- If class has *no constructors*
  - **Compiler** provides a default constructor
    - Sets all instance fields to their default values
- If a class has at least one constructor and no default constructor
  - Default constructor is NOT provided

Default Constructor

- Chicken class has one constructor:

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

➡ No default constructor

```
Chicken chicken = new Chicken();
```

- Is a compiler error

Constructors Calling Constructors

- Can call a constructor from inside another constructor
- The **first** statement of constructor must be

```
this( . . . );
```

to call another constructor of the same class

➢ `this` refers to the object being constructed

Why would you want to call another constructor?

Constructors Calling Constructors

- Why would you call another constructor?
  - Reduce code size/reduce duplicate code
- Ex: if name not provided, use default name

```
Chicken( int height, double weight ) {
      this( "Bubba", height, weight);
}
```

- Another example:

```
Chicken( int height, double weight ) {
      this();
      this.height = height;
      this.weight = weight;
}
```

Not in example code online