

Objectives

- Parameter passing
- Enforcing encapsulation: Cloning

Review

- What does `static` mean?
- Static methods:
 - What are static methods similar to in Python?
 - What does a static method have access to?
 - How do you call a static method?
 - When should we make a method static?
- When should we make a field static?
- How do you create pretty, formatted output?
 - What is the syntax? What are the components?

PARAMETER PASSING

Method Parameters in Java

- Java always passes parameters into methods ***by value***
 - Meaning: the formal parameter becomes a copy of the argument/actual parameter's value
 - caller and callee have two independent variables with the *same* value
 - Consequence: Methods **cannot** change the ***variables*** used as input parameters
 - A subtle point, so we will go through several examples
- Python is something that's not quite pass-by-value—it depends on if the object is mutable or immutable
 - *Pass-by-alias* is one term used

Method Parameters in Java

```
public static void main(String[] args) {  
    int x = 10;  
    int squared = square(x);  
    System.out.println("The square of " + x + " is " +  
                        squared);  
}  
  
public static int square(int num) {  
    return num*=num;  
}
```

Draw the stack as it changes
(similar to Python):

Method Parameters in Java

```
public static void main(String[] args) {  
    int x = 10;  
    int squared = square(x);  
    System.out.println("The square of " + x + " is " +  
                        squared);  
}  
  
public static int square(int num) {  
    return num*num;  
}
```

Stack:

main	x 10 squared
------	-----------------

Method Parameters in Java

```
public static void main(String[] args) {  
    int x = 10;  
    int squared = square(x);  
    System.out.println("The square of " + x + " is " +  
                        squared);  
}  
  
public static int square(int num) {  
    return num*num;  
}
```

num copies the value of x

square	num 10
main	x 10 squared

Method Parameters in Java

```
public static void main(String[] args) {  
    int x = 10;  
    int squared = square(x);  
    System.out.println("The square of " + x + " is " +  
                        squared);  
}  
  
public static int square(int num) {  
    return num*=num;  
}
```

square	num 100
main	x 10 squared

Method Parameters in Java

```
public static void main(String[] args) {  
    int x = 10;  
    int squared = square(x);  
    System.out.println("The square of " + x + " is " +  
                        squared);  
}  
  
public static int square(int num) {  
    return num*num;  
}
```

Output:

The square of 10 is 100

main	x 10 squared 100
------	---------------------

What's the Output?

```
public static void main(String[] args) {  
    int x = 27;  
    System.out.println(x);  
    doubleValue(x);  
    System.out.println(x);  
}  
public static void doubleValue(int p) {  
    p = p * 2;  
}
```

1. Think (independently) for 1 minute
2. Share with your neighbor.
3. Discuss as class

What's the Output?

```
public static void main(String[] args) {  
    int x = 27;  
    System.out.println(x);  
    doubleValue(x);  
    System.out.println(x);  
}  
public static void doubleValue(int p) {  
    p = p * 2;  
}
```

Output (so far):
27

double Value	p 27
main	x 27

What's the Output?

```
public static void main(String[] args) {  
    int x = 27;  
    System.out.println(x);  
    doubleValue(x);  
    System.out.println(x);  
}  
public static void doubleValue(int p) {  
    p = p * 2;  
}
```

Output (so far):
27

double Value	p 54
main	x 27

What's the Output?

```
public static void main(String[] args) {  
    int x = 27;  
    System.out.println(x);  
    doubleValue(x);  
    System.out.println(x);  
}  
public static void doubleValue(int p) {  
    p = p * 2;  
}
```

Output:

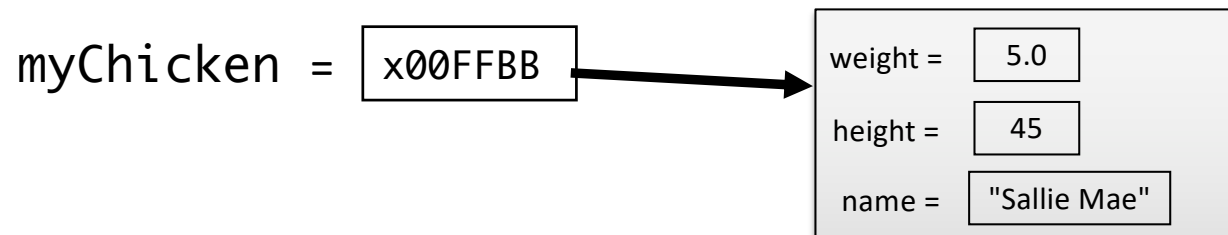
27
27

main	x 27
------	------

Recall: *Object* variables = memory location

- myChicken is a Chicken variable

- Contains the *location* of where the Chicken object is stored in memory



Pass by Value: Objects

- Pass Chicken object variable to methodName:

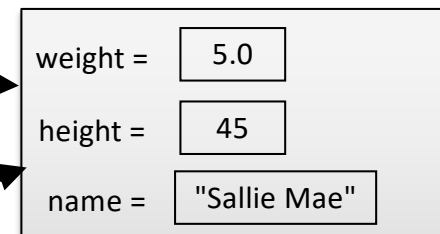
```
methodName(myChicken)
```

```
public void methodName(Chicken c)
```

Actual parameter: myChicken = x00FFBB

c copies value of myChicken

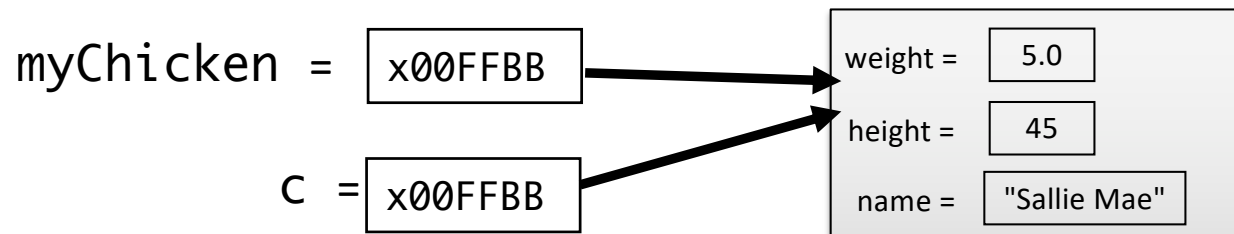
Formal parameter: c = x00FFBB



Pass by Value: Objects

- What happens in this case?

```
methodName(myChicken);
```



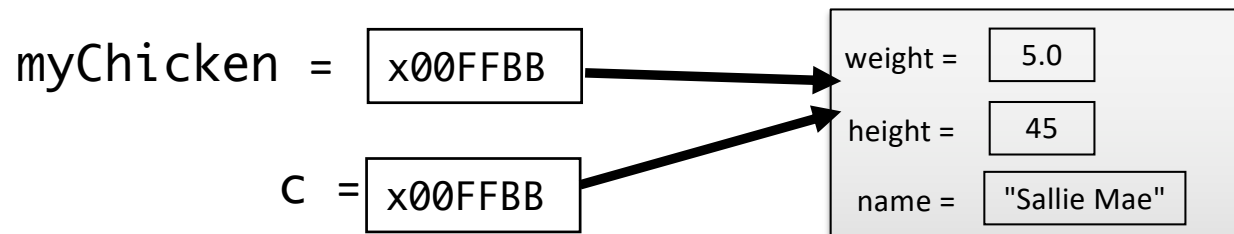
```
public void methodName(Chicken c) {  
    if( c.getWeight() < MIN ) {  
        c.feed();  
    }  
}
```

Could myChicken be changed when the called method is executed?

Pass by Value: Objects

- What happens in this case?

```
methodName(myChicken);
```



```
public void methodName(Chicken c) {  
    if( c.getWeight() < MIN ) {  
        c.feed();  
    }  
}
```

Can the Chicken object `myChicken` change in the called method?

YES! Both `chicken` and `c` are pointing to the *same* Chicken object

Example 1: What's the Output?

```
// From Farm class
public static void main(String[] args) {
    Farm farm = new Farm("OldMac");
    Chicken sal = new Chicken("Sallie Mae", 5.0, 45);
    System.out.println(sal.getWeight());
    farm.feedChicken(sal);
    System.out.println(sal.getWeight());
}

public void feedChicken(Chicken c) {
    c.setWeight( c.getWeight() + .5);
}
```

(setWeight was not a method defined in our Chicken class; just for this example)

Example 1: What's the Output?

```
// From Farm class
public static void main(String[] args) {
    Farm farm = new Farm("OldMac");
    Chicken sal = new Chicken("Sallie Mae", 5.0, 45);
    System.out.println(sal.getWeight());
    farm.feedChicken(sal);
    System.out.println(sal.getWeight());
}

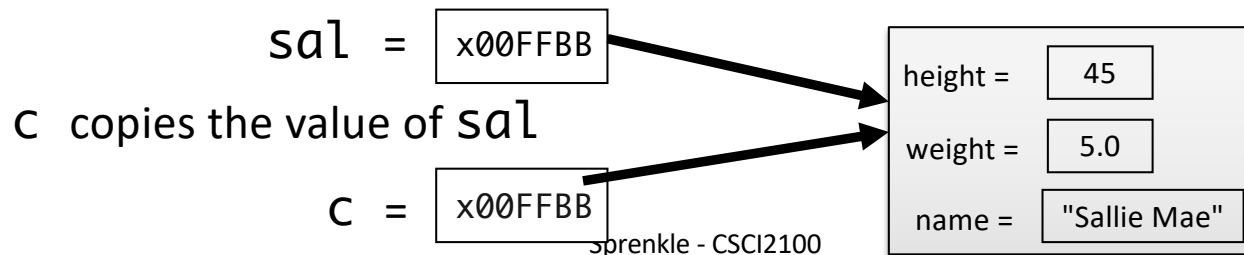
public void feedChicken(Chicken c) {
    c.setWeight( c.getWeight() + .5);
}
```



Example 1: What's the Output?

```
// From Farm class
public static void main(String[] args) {
    Farm farm = new Farm("OldMac");
    Chicken sal = new Chicken("Sallie Mae", 5.0, 45);
    System.out.println(sal.getWeight());
    farm.feedChicken(sal);
    System.out.println(sal.getWeight());
}

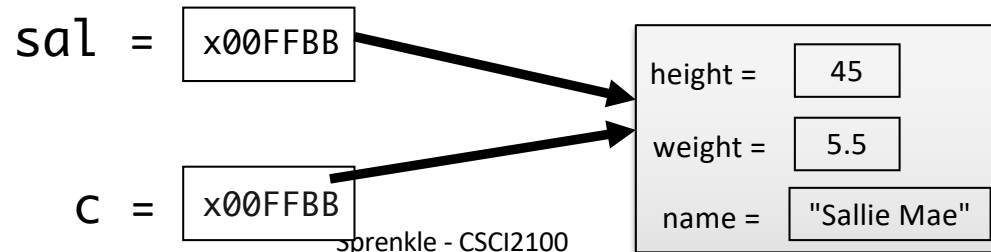
public void feedChicken(Chicken c) {
    c.setWeight( c.getWeight() + .5);
}
```



Example 1: What's the Output?

```
// From Farm class
public static void main(String[] args) {
    Farm farm = new Farm("OldMac");
    Chicken sal = new Chicken("Sallie Mae", 5.0, 45);
    System.out.println(sal.getWeight());
    farm.feedChicken(sal);
    System.out.println(sal.getWeight());
}

public void feedChicken(Chicken c) {
    c.setWeight( c.getWeight() + .5);
}
```



Example 1: What's the Output?

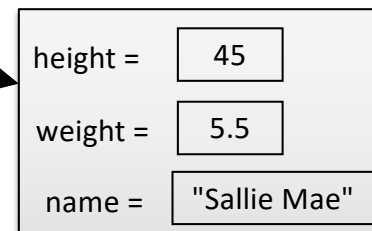
Note: feedChicken
could be static;
doesn't make a
difference in output

```
// From Farm class
public static void main(String[] args) {
    Farm farm = new Farm("OldMac");
    Chicken sal = new Chicken("Sallie Mae", 5.0, 45);
    System.out.println(sal.getWeight());
    farm.feedChicken(sal);
    System.out.println(sal.getWeight());
}

public void feedChicken(Chicken c) {
    c.setWeight( c.getWeight() + .5);
}
```

5.0
5.5

sal = x00FFBB

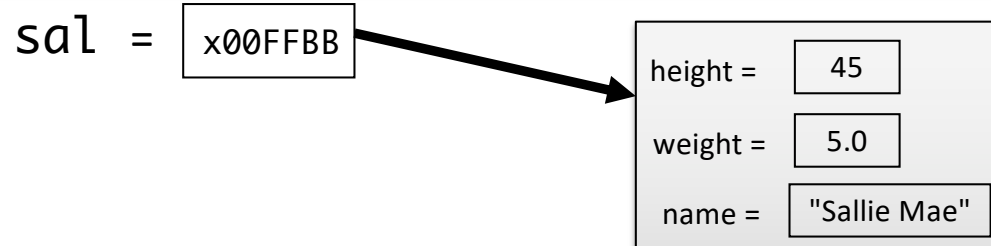


Example 2: What's the Output?

```
public static void main(String[] args) {  
    Farm farm = new Farm("OldMac");  
    Chicken sal = new Chicken("Sallie Mae", 5.0, 45);  
    System.out.println(sal.getWeight());  
    farm.feedChicken(sal);  
    System.out.println(sal.getWeight());  
}  
  
public void feedChicken(Chicken c) {  
    c = new Chicken(c.getName(), c.getWeight(), c.getHeight() );  
    c.setWeight( c.getWeight() + .5);  
}
```

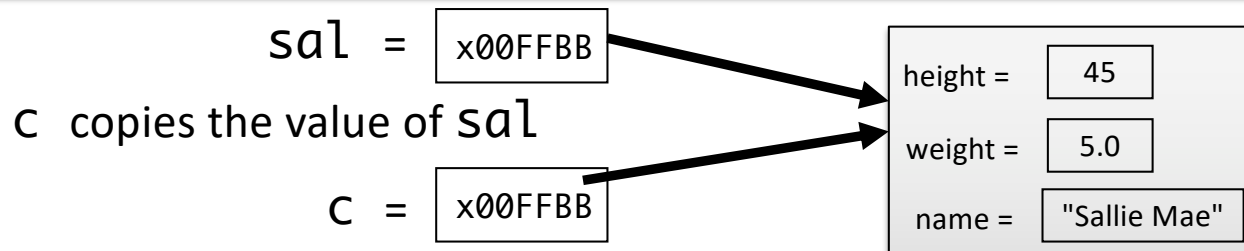
Example 2: Tracing through Execution

```
public static void main(String[] args) {  
    Farm farm = new Farm("OldMac");  
    Chicken sal = new Chicken("Sallie Mae", 5.0, 45);  
    System.out.println(sal.getWeight());  
    farm.feedChicken(sal);  
    System.out.println(sal.getWeight());  
}  
  
public void feedChicken(Chicken c) {  
    c = new Chicken(c.getName(), c.getWeight(), c.getHeight() );  
    c.setWeight( c.getWeight() + .5);  
}
```



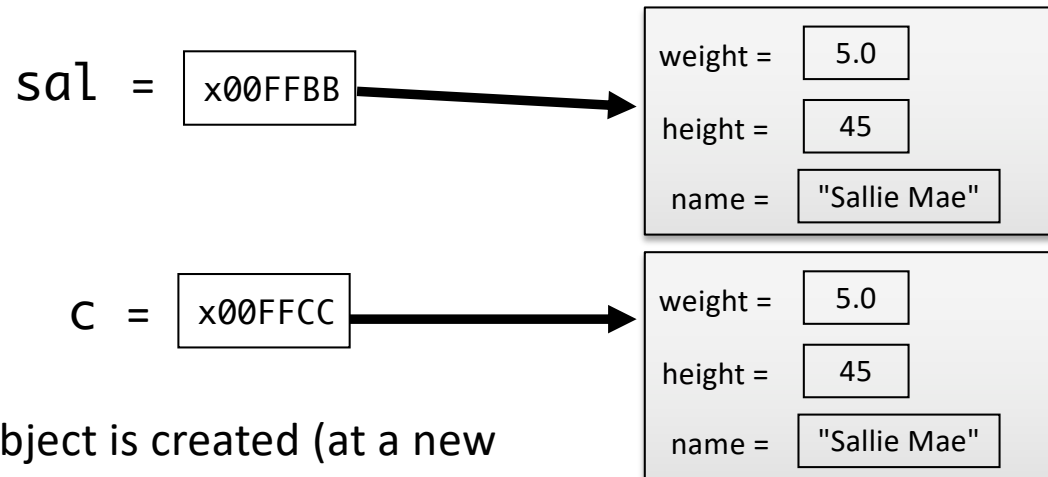
Example 2: Tracing through Execution

```
public static void main(String[] args) {  
    Farm farm = new Farm("OldMac");  
    Chicken sal = new Chicken("Sallie Mae", 5.0, 45);  
    System.out.println(sal.getWeight());  
    farm.feedChicken(sal);  
    System.out.println(sal.getWeight());  
}  
  
public void feedChicken(Chicken c) {  
    c = new Chicken(c.getName(), c.getWeight(), c.getHeight() );  
    c.setWeight( c.getWeight() + .5);  
}
```



Example 2: Tracing through Execution

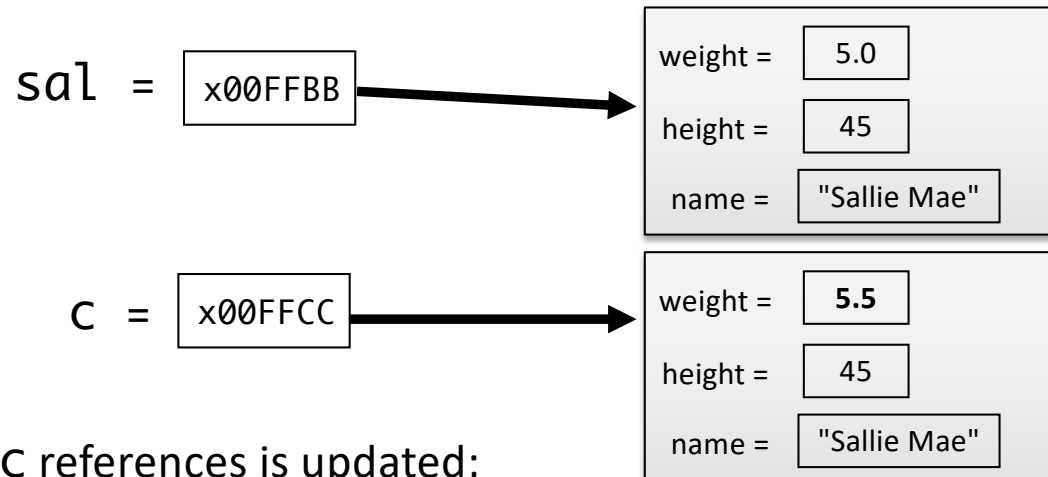
```
public void feedChicken(Chicken c) {  
    c = new Chicken(c.getName(), c.getWeight(), c.getHeight() );  
    c.setWeight( c.getWeight() + .5);  
}
```



A new `Chicken` object is created (at a new memory address). `c` is assigned to/ references the new object. The *variable* `c` has changed → no longer has the same value as `sal`.

Example 2: Tracing through Execution

```
public void feedChicken(Chicken c) {  
    c = new Chicken(c.getName(), c.getWeight(), c.getHeight() );  
    c.setWeight( c.getWeight() + .5);  
}
```



The object that `c` references is updated;
the object that `sal` references is unaffected

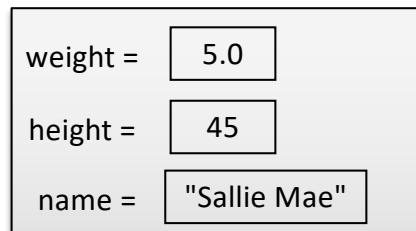
Example 2: Tracing through Execution

```
public static void main(String[] args) {  
    Farm farm = new Farm("OldMac");  
    Chicken sal = new Chicken("Sallie Mae", 5.0, 45);  
    System.out.println(sal.getWeight());  
    farm.feedChicken(sal);  
    System.out.println(sal.getWeight());  
}  
  
public void feedChicken(Chicken c) {  
    c = new Chicken(c.getName(), c.getWeight(), c.getHeight() );  
    c.setWeight( c.getWeight() + .5);  
}
```

5.0
5.0

sal =

x00FFBB



c is out of scope/no longer accessible

Summary of Passing Parameters to Methods

- Parameters are passed **by value** in Java
- An **object variable** (not an object) is passed into a method
 - Called method does **not** get a copy of the original object but a copy of the *memory location* of the object
 - Changing the *state* of an object in a method changes the state of object outside the method

ENFORCING ENCAPSULATION

Encapsulation/Black-Box Programming Revisited

- Objects should hide their data
- Only allow other objects to access this data through **accessor** and **mutator** methods

Example of Black-box Programming

```
private int height; // in cm
private double weight; // in lbs
...

public void feed() {
    weight += .3;
    height += 1;
}
```

- We don't want to allow direct changing of Chicken's state (height and weight)
 - Don't want them set to 0 or negative values
 - We want the height and weight to be proportional, so no separate setHeight and setWeight methods
- Only allowing access to the methods allows us to restrict the kinds of changes that can be made to the state of the object

Encapsulation/Black-Box Programming Revisited

- Objects should hide their data
- Only allow other objects to access this data through **accessor** and **mutator** methods
- Common programmer mistake:
 - Creating an accessor method that returns a reference to a mutable (changeable) object

Violating Black-Box Programming Principle

```
public class Farm {  
    . . .  
    private Chicken headRooster;  
  
    public Chicken getHeadRooster() {  
        return headRooster;  
    }  
    . . .  
}
```

Why/how does this code violate black-box programming principle?

Violating Black-Box Programming Principle

```
public class Farm {  
    . . .  
    private Chicken headRooster;  
  
    public Chicken getHeadRooster() {  
        return this.headRooster;  
    }  
    . . .  
}
```

Problem: Gives others access to Farm's headRooster through the *public* getHeadRooster method. Others can then feed your rooster or change his name!! (Silly example; understand consequences)

```
public class BadGuyCode {  
    . . .  
    Chicken stolen = farm.getHeadRooster();  
    System.out.println("mwha hahaha");  
    stolen.setName("Dinner");  
    ...  
}
```

Fixing the Problem: Cloning

```
public class Farm {  
    . . .  
    private Chicken headRooster;  
  
    public Chicken getHeadRooster() {  
        return (Chicken) headRooster.clone();  
    }  
    . . .  
}
```

Method is available to all objects
(inherited from Object)

- In previous example, could modify returned object's state
- Here, another Chicken object, with the same data as headRooster, is created and returned to the user
- If the user modifies (e.g., feeds) that object, Farm's headRooster is not affected

Cloning

- Cloning is a more complicated topic than it seems from the example
 - Out of scope for this class
- See *Effective Java* for more information

What about the Chicken's getter methods?

```
public class Chicken {  
    private int height;  
    private String name;  
    . . .  
  
    public String getName() {  
        return this.name;  
    }  
  
    public int getHeight() {  
        return this.height;  
    }  
  
    .  
}
```

Why was it okay to return the name, height, or weight of a chicken in public methods?

What about the Chicken's getter methods?

```
public class Chicken {  
    private int height;  
    private String name;  
    . . .  
  
    public String getName() {  
        return this.name;  
    }  
  
    public int getHeight() {  
        return this.height;  
    }  
    . . .  
}
```

Similar to Python, primitive types and Strings are *immutable*. Since those attributes have immutable data types (String, int, double, respectively), others can't change those attributes when retrieved using a getter method.

Looking Ahead: Assignment 2

- Due tonight at 11:59
- Lots of flexibility in design in Birthday and BirthdayParadox
 - BirthdayParadox **should use/leverage** Birthday class
- Lots of different correct designs
 - BUT many more incorrect or too-complicated designs
- Consider
 - If a variable should be a local variable, instance variable, or class variable
 - Rule of thumb: make a local variable and then can change later.
 - How can I break this into smaller problems? → Methods!
 - API for the methods: What is its input? What is its output (what is returned)?
- Test small parts!
- Use git well
 - When are good points to checkpoint (commit) or make a new branch?

Looking Ahead: Exam 1

- Exam 1 – Friday

- Timed exam, in-class, on paper: 60 minutes

- 1 letter-size page of notes for reference

- Prep document online

- 3 sections:

- Very Short Answer, Short Answer, Coding

Example 1: Changed feedChicken to static

```
// From Farm class
public static void main(String[] args) {
    Farm farm = new Farm("OldMac");
    Chicken sal = new Chicken("Sallie Mae", 5.0, 45);
    System.out.println(sal.getWeight());
    feedChicken(sal);
    System.out.println(sal.getWeight());
}

public static void feedChicken(Chicken c) {
    c.setWeight( c.getWeight() + .5);
}
```

Does not change results/outcome

(setWeight was not a method defined in our Chicken class; just for this example)

Example 2: Changed feedChicken to static

```
public static void main(String[] args) {  
    Farm farm = new Farm("OldMac");  
    Chicken sal = new Chicken("Sallie Mae", 5.0, 45);  
    System.out.println(sal.getWeight());  
    feedChicken(sal);  
    System.out.println(sal.getWeight());  
}  
  
public static void feedChicken(Chicken c) {  
    c = new Chicken(c.getName(), c.getWeight(), c.getHeight() );  
    c.setWeight( c.getWeight() + .5);  
}
```

Does not change results/outcome