Objectives

- Formatting
- Enforcing encapsulation: Cloning
- Parameter passing
- Garbage collection

1

Assignment Feedback

- Why articulation of errors matters
  - Demonstrates your understanding (or lack of understanding)
  - You will need to discuss coding with teammates
- Why output files matter
  - I can see if when you ran on your machine, you get the same output I get

2

Assignment 4

- Lots of flexibility in design in `Birthday` and `BirthdayParadox`
- Lots of different correct designs
  - Many more incorrect or too-complicated designs
- Consider
  - If a variable should be a local variable, instance variable, or class variable
  - 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?

Review

- What is overloading?
  - Why would we want to overload a method?
- What is overriding?
- How do we make an instance variable unchangeable after construction?
- How do we call a constructor within a constructor?
- What is the root of the Java class hierarchy?
- What method should we implement to allow pretty printing of objects we define?
- What method should we implement for determining if two objects are equivalent?

FORMATTING

5

Formatting Strings: `format`

- `String.format(<templatestring>, <value1>, <value2>, …, <valuen>)`

  Replacement values

- Semantics: creates, returns a **formatted string**
  - Means "format the templatestring, using the format(s) specified by **format specifiers** on the corresponding replacement values"
- Typically used with print statements

6

Formatting Strings

- **`templatestring`** is a template for the resulting string with format specifiers instead of the values
  - For each format specifier in templatestring, should have a ***replacement value***

```
String.format("%.2f", 3.14159 );
```

result is `"3.14"`

One format specifier

Corresponding replacement value

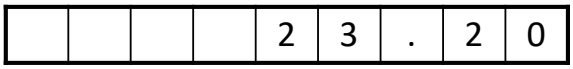
Example Format Specifiers

`String.format("%5d", 12)`

Field width is 5

Right-justified

`String.format("%9.2f", 23.1999)`



Precision is 2

Field width is 9

- What if precision is bigger than the decimal places?
  - Fills decimal with 0s
- What if field width is smaller than the length of the value?
  - String contains entire value

Format Specifiers

[] mean "optional"

- General format:
  `%[flags][width][.precision]conversion`
  - flags:
    - 0: zero fills
    - +: adds a + sign before positive values
    - -: left-justification (default is right-justification)
  - width:
    - *Minimum* number of character spaces reserved to display the entire value
    - Includes decimal point, digits before and after the decimal point and the sign

Format Specifiers

- General format:
  `%[flags][width][.precision]conversion`
  - precision:
    - Number of digits after the decimal point for **floating point** values
  - code:
    - Indicates the value's **type/**way to format

Conversion	Type
s	string
d	integer
f	float/double

Partial Examples using format

Format specifier

```
String.format("Your item that cost ($%.2f)", value);
String.format("costs $%.2f with tax", tax);
```

Replacement values

Common use case:
Save each of these in two (String) variables and print them

Example: Printing Out Tables

- A table of temperature conversions

```
Temp F      Temp C      Temp K
------      ------      ------
-459.7      -273.1         0.0
   0.0       -17.8       255.4
  32.0         0.0       273.1
```

- If we want to print data in rows, what is the template for what a row looks like?
  - How do we make the column labels line up?

 TemperatureTable.java

Example: Printing Out Tables

Using `String.format`

```
// example for one line of data in the table
double[] temps = {-459.7, -273.1, 0.0};

String tempFormat = "%10.1f %10.1f %10.1f";

System.out.println(String.format(tempFormat,
      temps[0], temps[1], temps[2]));
```


Example: Printing Out Tables

Using `System.out.printf`

```
// example for one line of data in the table
double[] temps = {-459.7, -273.1, 0.0};

String tempFormat = "%10.1f %10.1f %10.1f\n";

System.out.printf(tempFormat,
      temps[0], temps[1], temps[2]);
```

ENFORCING ENCAPSULATION

15

Encapsulation/Black-Box Programming Revisited

- Objects should hide their data and 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

16

What is "bad" about this class?

```
public class Farm {
        . . .
        private Chicken headRooster;

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


What is "bad" about this class?

```
public class Farm {
        . . .
        private Chicken headRooster;

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

Problem: Giving others access to Farm's headRooster
Others can then feed your rooster or change his name!!
(Silly example; understand consequences)

```
public class OtherCode {
        . . .
        Chicken stolen = farm.getHeadRooster();
        . . .
}
```

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
- 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, `headRooster` is not affected

Cloning

- Cloning is a more complicated topic than it seems from the example
  - Out of scope for this class

What is "bad" about this class?

```
public class Farm {
        . . .
        private Chicken headRooster;

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

Problem: Giving others access to Farm's headRooster
Others can then feed your rooster or change his name!!
(Silly example; understand consequences)

But, then, why is it okay to return the name, height, or weight of a chicken?
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.

PARAMETER PASSING

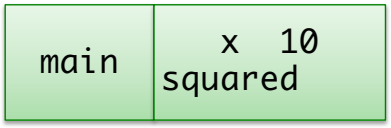
Method Parameters in Java

- 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 can**not** 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;
}
```

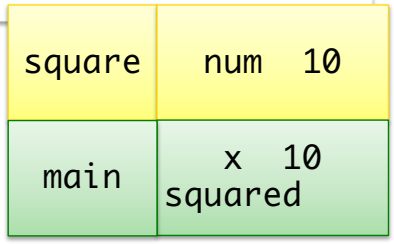
Draw the stack as it changes (similar to Python):



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;
}
```

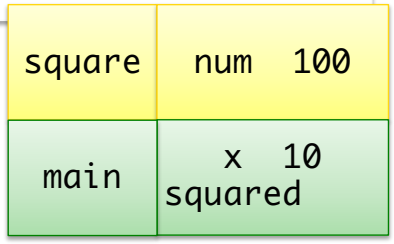
num copies the value of x



25

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;
}
```



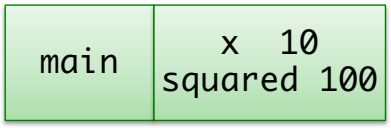
26

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
```



What's the Output?

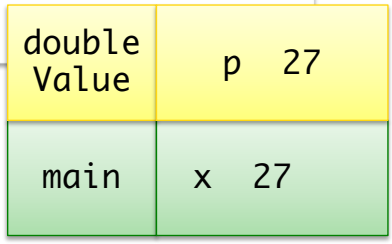
```
public static void main(String[] args) {
        int x = 27;
        System.out.println(x);
        doubleValue(x);
        System.out.println(x);
}
public static void doubleValue(int p) {
        p = p * 2;
}
```

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;
}
```

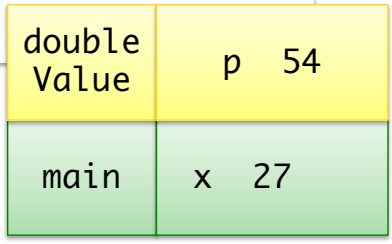
Output (so far):
27



29

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;
}
```

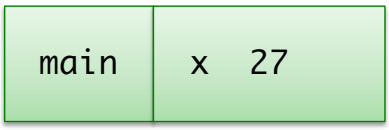


30

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;
}
```

```
27
27
```



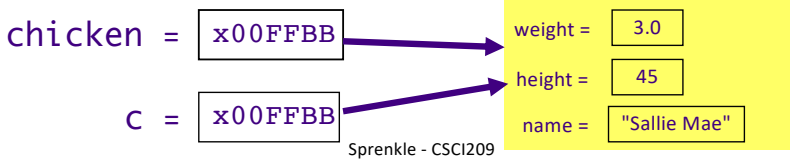
Pass by Value: Objects

- Primitive types are a little more obvious
  - Can't change original variable
- For objects, passing a copy of the parameter looks like:

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

Pass Chicken object to `methodName` when calling method

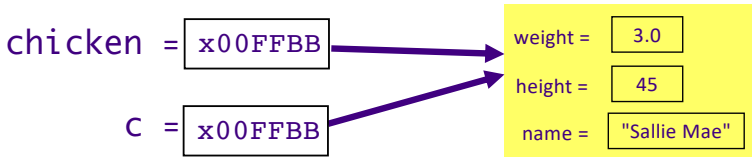
```
methodName(chicken);
```



Pass by Value: Objects

- What happens in this case?

```
methodName(chicken);
```



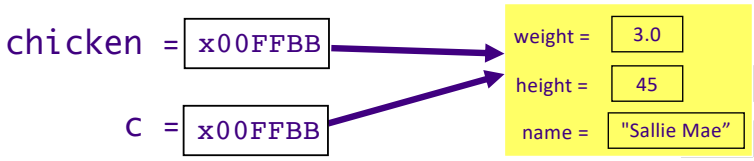

```
public void methodName(Chicken c) {
        if( c.getWeight() < MIN ) {
                c.feed();
        }
        …
}
```

Can the Chicken object be changed in the called method?

Pass by Value: Objects

- What happens in this case?

```
methodName(chicken);
```




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

Can the Chicken object be changed in the called method?

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

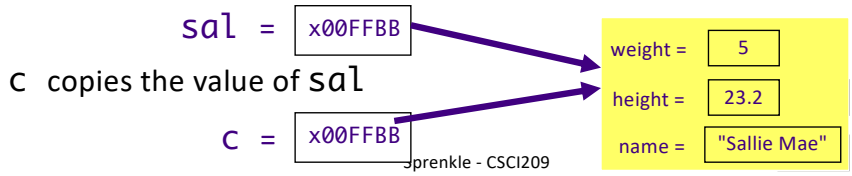
Example 1: What's the Output?

```
Farm farm = new Farm("OldMac");
Chicken sal = new Chicken("Sallie Mae", 5, 23.2);
System.out.println(sal.getWeight());
farm.feedChicken(sal);
System.out.println(sal.getWeight());
. . .

// From Farm class
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?

```
Farm farm = new Farm("OldMac");
Chicken sal = new Chicken("Sallie Mae", 5, 23.2);
System.out.println(sal.getWeight());
farm.feedChicken(sal);
System.out.println(sal.getWeight());
. . .

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



Example 1: What's the Output?

```
Farm farm = new Farm("OldMac");
Chicken sal = new Chicken("Sallie Mae", 5, 23.2);
System.out.println(sal.getWeight());
farm.feedChicken(sal);
System.out.println(sal.getWeight());
. . .

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

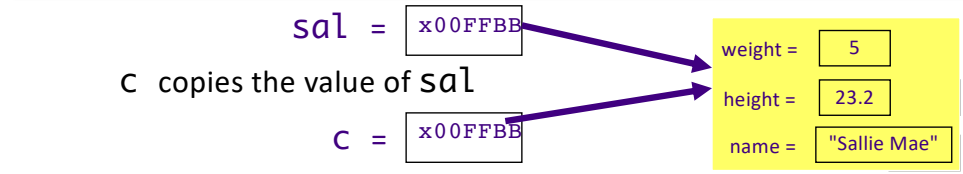
```
23.2
23.7
```


Example 2: What's the Output?

```
Farm farm = new Farm("OldMac");
Chicken sal = new Chicken("Sallie Mae", 5, 23.2);
System.out.println(sal.getWeight());
farm.feedChicken(sal);
System.out.println(sal.getWeight());
. . .

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

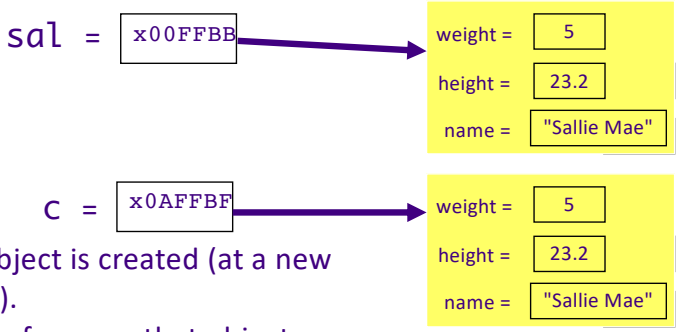
Example 2: Tracing through Execution

```
Farm farm = new Farm("OldMac");
Chicken sal = new Chicken("Sallie Mae", 5, 23.2);
System.out.println(sal.getWeight());
farm.feedChicken(sal);
System.out.println(sal.getWeight());
. . .
// From Farm class
public void feedChicken(Chicken c) {
    c = new Chicken(c.getName(), c.getWeight(), c.getHeight() );
    c.setWeight( c.getWeight() + .5);
}
```



Example 2: Tracing through Execution

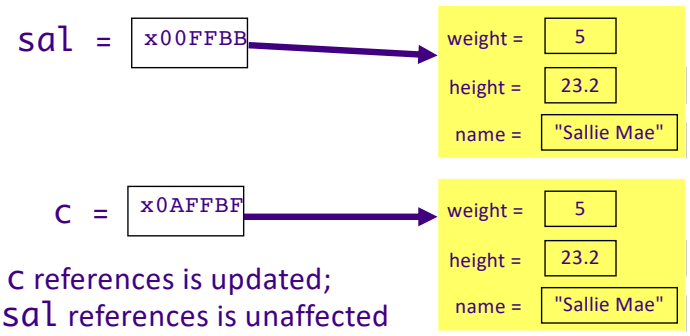
```
public void feedChicken(Chicken c) {
    c = new Chicken(c.getName(), c.getWeight(), c.getHeight() );
    c.setWeight( c.getWeight() + .5);
}
```



A new Chicken object is created (at a new memory address).
c is assigned to/references that object.

Example 2: Tracing through Execution

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

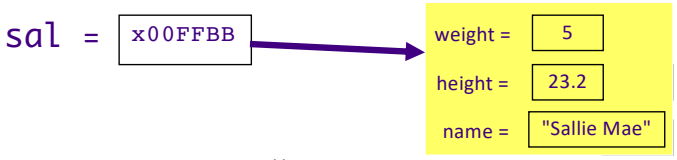


The object that c references is updated;
the object that sal references is unaffected

Example 2: Tracing through Execution

```
Farm farm = new Farm("OldMac");
Chicken sal = new Chicken("Sallie Mae", 5, 23.2);
System.out.println(sal.getWeight());
farm.feedChicken(sal);
System.out.println(sal.getWeight());
. . .
// From Farm class
public void feedChicken(Chicken c) {
        c = new Chicken(c.getName(), c.getWeight(),
                c.getHeight() );
        c.setWeight( c.getWeight() + .5);
}
```

```
23.2
23.2
```



Summary of Passing Parameters to Methods

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

TRACING THROUGH CODE

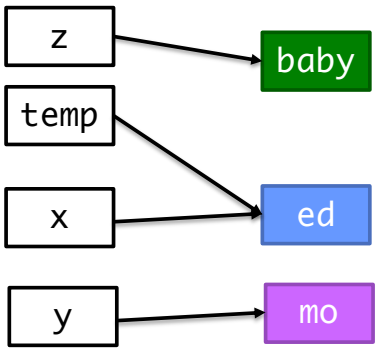
What Happens in This Code?

```
Chicken x, y;
Chicken z = new Chicken("baby", 5, 1.0);
x = new Chicken("ed", 81, 10.3);
y = new Chicken("mo", 63, 6.2);
Chicken temp = x;
x = y;
y = temp;
z = x;
```

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

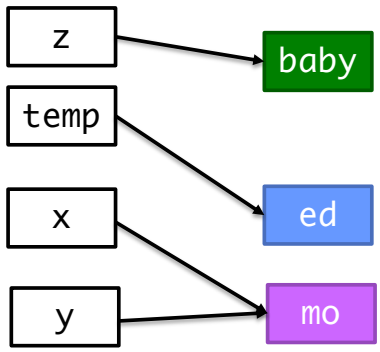
What Happens in This Code?

```
Chicken x, y;
Chicken z = new Chicken("baby", 5, 1.0);
x = new Chicken("ed", 81, 10.3);
y = new Chicken("mo", 63, 6.2);
Chicken temp = x;
x = y;
y = temp;
z = x;
```



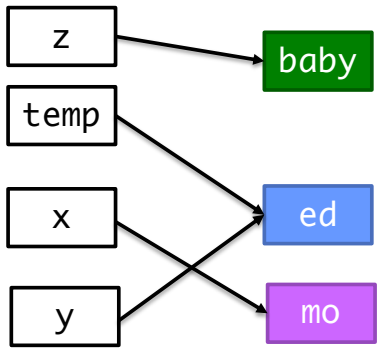
What Happens in This Code?

```
Chicken x, y;
Chicken z = new Chicken("baby", 5, 1.0);
x = new Chicken("ed", 81, 10.3);
y = new Chicken("mo", 63, 6.2);
Chicken temp = x;
x = y;
y = temp;
z = x;
```



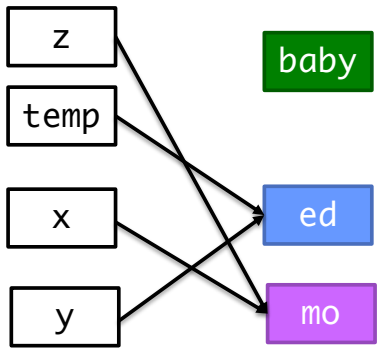
What Happens in This Code?

```
Chicken x, y;
Chicken z = new Chicken("baby", 5, 1.0);
x = new Chicken("ed", 81, 10.3);
y = new Chicken("mo", 63, 6.2);
Chicken temp = x;
x = y;
y = temp;
z = x;
```



What Happens in This Code?

```
Chicken x, y;
Chicken z = new Chicken("baby", 5, 1.0);
x = new Chicken("ed", 81, 10.3);
y = new Chicken("mo", 63, 6.2);
Chicken temp = x;
x = y;
y = temp;
z = x;
```



What Happens in This Code?

```
Chicken x, y;
Chicken z = new Chicken("baby", 5, 1.0);
x = new Chicken("ed", 81, 10.3);
y = new Chicken("mo", 63, 6.2);
Chicken temp = x;
x = y;
y = temp;
z = x;
```

Whoops! Lost "baby" chicken! -- No object variable references it
Memory leak!

Luckily Java has ***garbage collectors*** to clean up the memory leak

Looking Ahead

- Assignment 4 – due Tuesday at 11:59 p.m.
  - Building on the Birthday class
    - Overloading constructor
    - Overriding methods
  - Creating an application, practicing
    - Control structures
    - Using your own class and classes from the Java API