Objectives

- Testing
- Enforcing encapsulation: Cloning
- Parameter passing

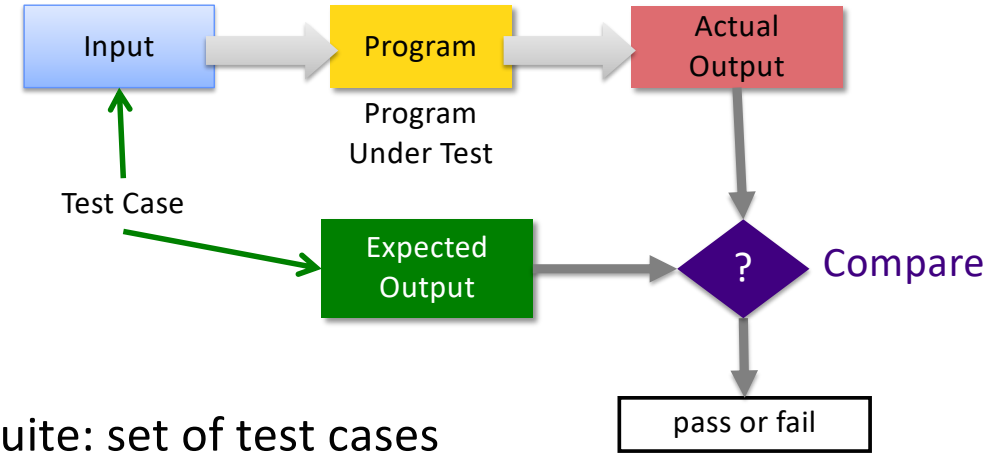
1

Review

- What does `static` mean?
- 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?

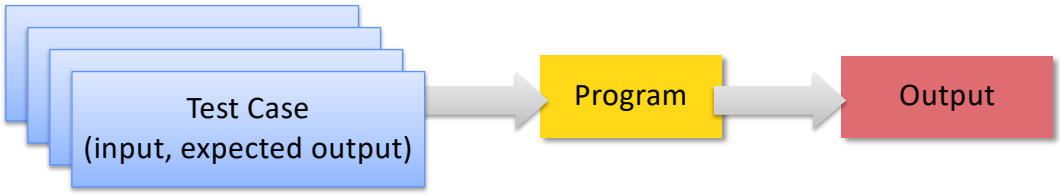
2

Software Testing Process



- Test Suite: set of test cases

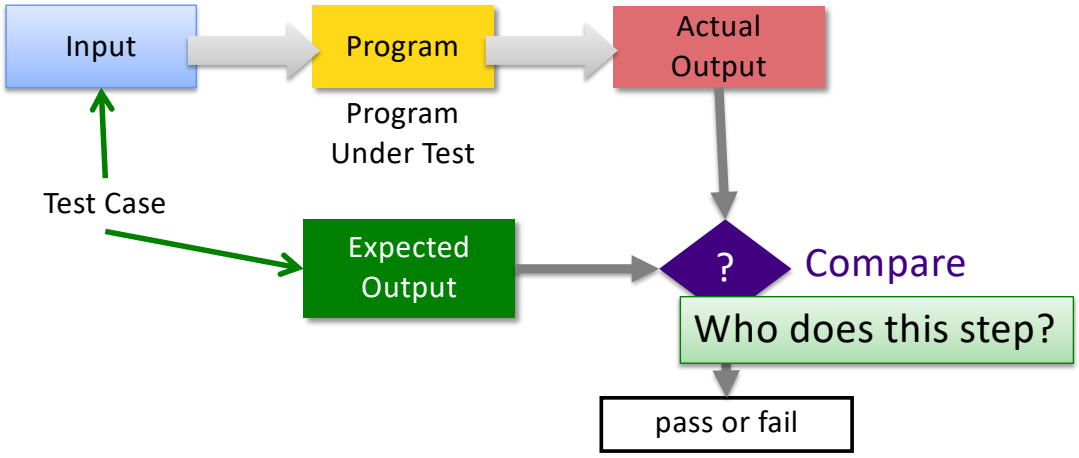
Software Testing Process



- Tester plays devil's advocate
  - ***Hopes*** to reveal problems in the program using "good" test cases
  - Better tester finds than a customer!

How is **testing** different from **debugging**?

Software Testing Process



Our Tests in `main`

- Execute the code to expose errors, automatically/programmatically
  - Let the *code* make the comparison instead of us
- Compare what we expect to the actual value
  - For mutators, this usually requires two steps:
    - Execute the method
    - Use other methods to check that the appropriate mutation occurred
- If actual and expected are different, displays an error message

ENFORCING ENCAPSULATION

7

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

8

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: Why was 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

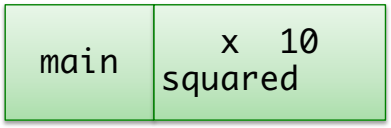
Method Parameters in Java

- 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;
}
```

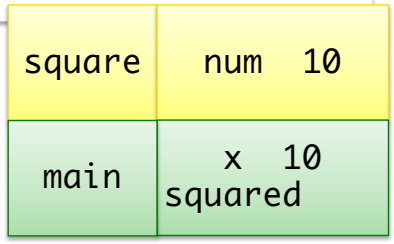
Draw the stack as it changes (similar to Python):



Method Parameters in Java

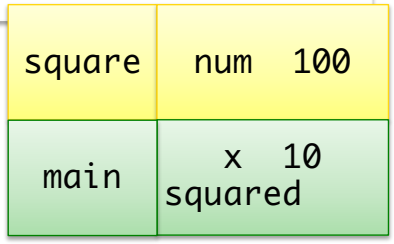
```
public static void main(String[] args) {
   int x = 10;
   int squared = square(x);
   System.out.println("The square of " + x + " is " +
                      squared);
}

public static int square(int num) {
   return num*=num;
}
```

num copies the value of x



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;
}
```

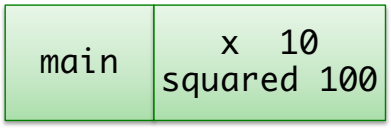


Method Parameters in Java

```
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?

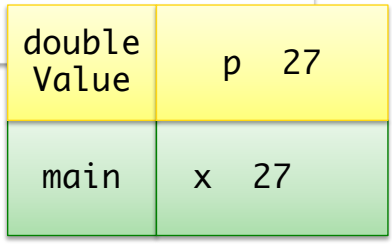
```
public static void main(String[] args) {
        int x = 27;
        System.out.println(x);
        doubleValue(x);
        System.out.println(x);
}
public static void doubleValue(int p) {
        p = p * 2;
}
```

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

What's the Output?

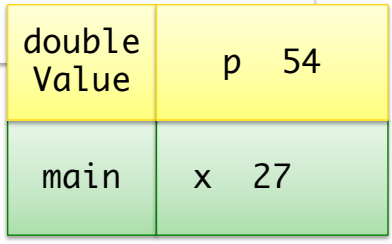
```
public static void main(String[] args) {
        int x = 27;
        System.out.println(x);
        doubleValue(x);
        System.out.println(x);
}
public static void doubleValue(int p) {
        p = p * 2;
}
```

Output (so far):
27



What's the Output?

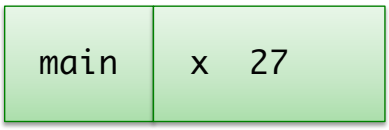
```
public static void main(String[] args) {
        int x = 27;
        System.out.println(x);
        doubleValue(x);
        System.out.println(x);
}
public static void doubleValue(int p) {
        p = p * 2;
}
```



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
```



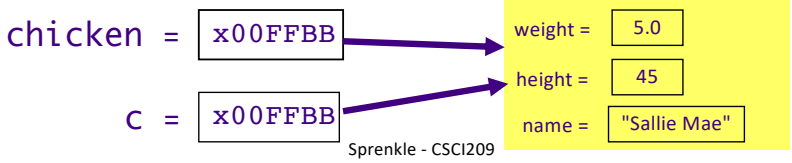
Pass by Value: Objects

- 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

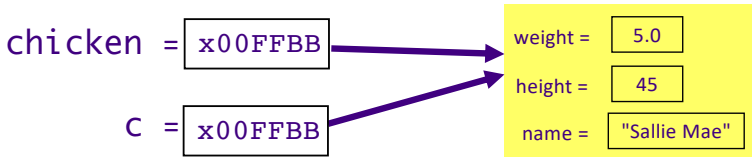
```
methodName(chicken);
```



Pass by Value: Objects

- What happens in this case?

```
methodName(chicken);
```



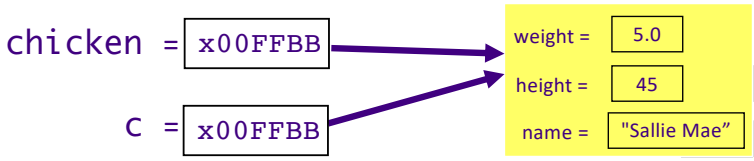

```
public void methodName(Chicken c) {
        if( c.getWeight() < MIN ) {
                c.feed();
        }
        …
}
```

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

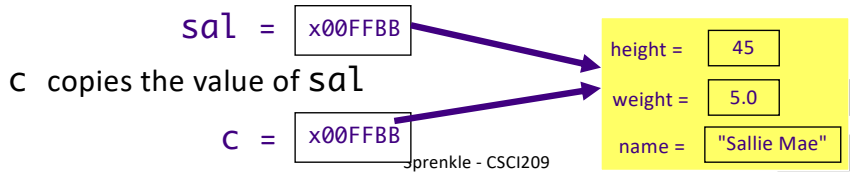
Example 1: What's the Output?

```
Farm farm = new Farm("OldMac");
Chicken sal = new Chicken("Sallie Mae", 5.0, 45);
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", 5.0, 45);
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", 5.0, 45);
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);
}
```

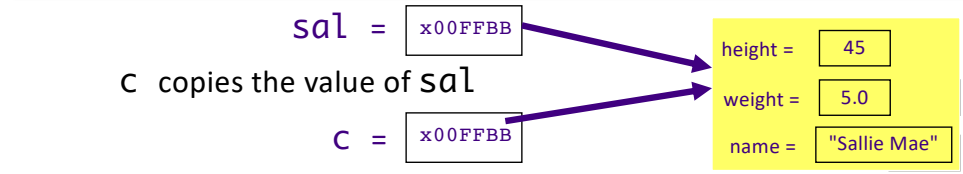
```
5.0
5.5
```


Example 2: What's the Output?

```
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());
. . .

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

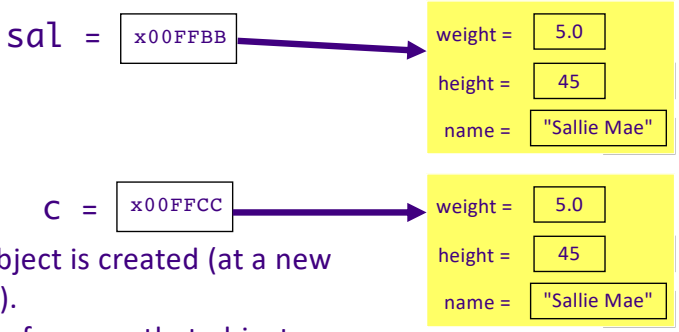
Example 2: Tracing through Execution

```
Farm farm = new Farm("OldMac");
Chicken sal = new Chicken("Sallie Mae", 5.0, 45);
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.getHeight(), c.getWeight() );
    c.setWeight( c.getWeight() + .5);
}
```



Example 2: Tracing through Execution

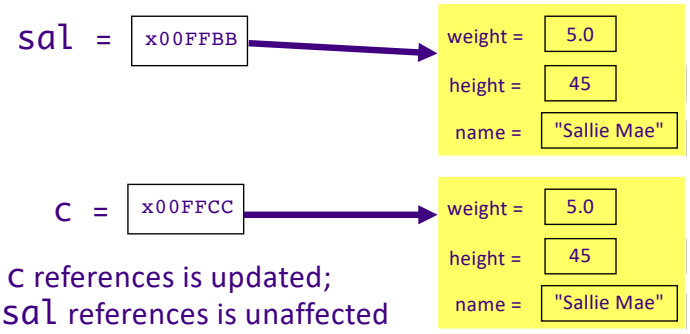
```
public void feedChicken(Chicken c) {
    c = new Chicken(c.getName(), c.getWeight(), c.getHeight() );
    c.setWeight( c.getWeight() + .5);
}
```



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

Example 2: Tracing through Execution

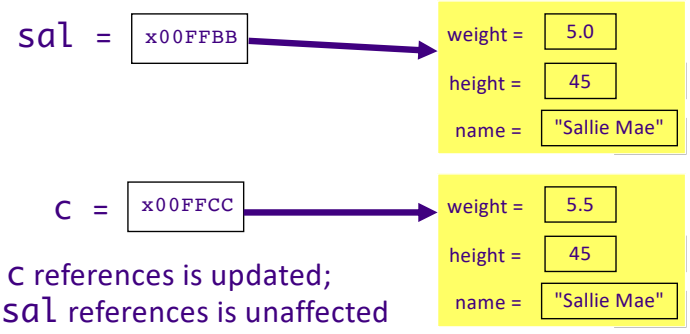
```
public void feedChicken(Chicken c) {
    c = new Chicken(c.getName(), c.getWeight(), c.getHeight() );
    c.setWeight( c.getWeight() + .5);
}
```



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

Example 2: Tracing through Execution

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

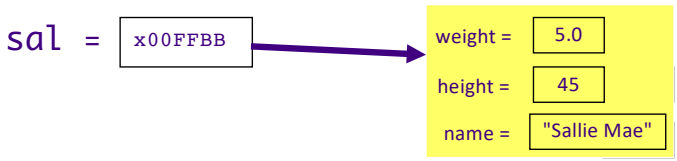


The object that c references is updated;
the object that sal references is unaffected

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

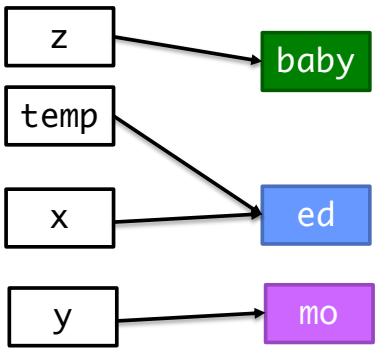
What Happens in This Code?

```
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
   - Draw it!
2. Share with your neighbor.
3. Discuss as class

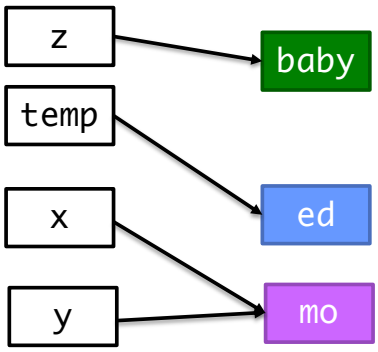
What Happens in This Code?

```
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;
```



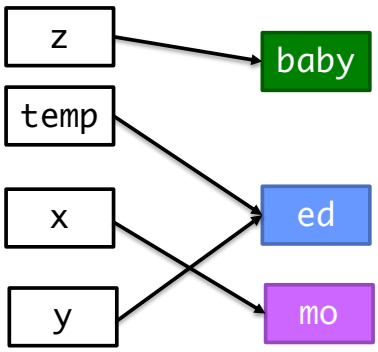
What Happens in This Code?

```
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;
```



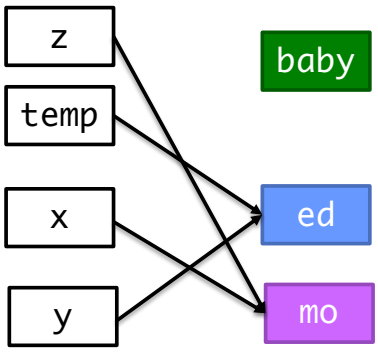
What Happens in This Code?

```
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 3 – due Wednesday before class
  - 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
- Exam 1 - Friday