Objectives

- Garbage Collection

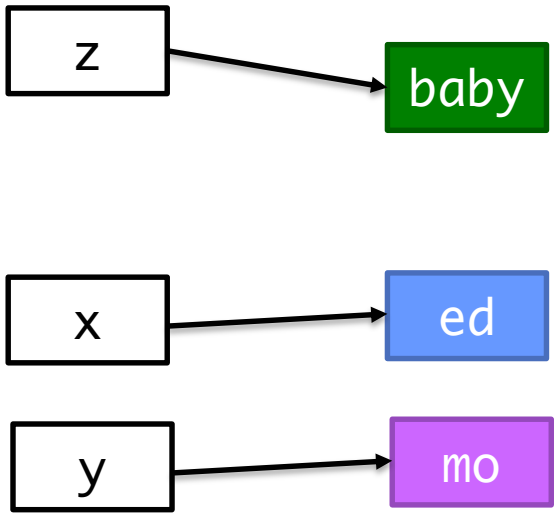
What Happens in This Code?

```
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;
```

Draw the code,
i.e., draw the objects and variables

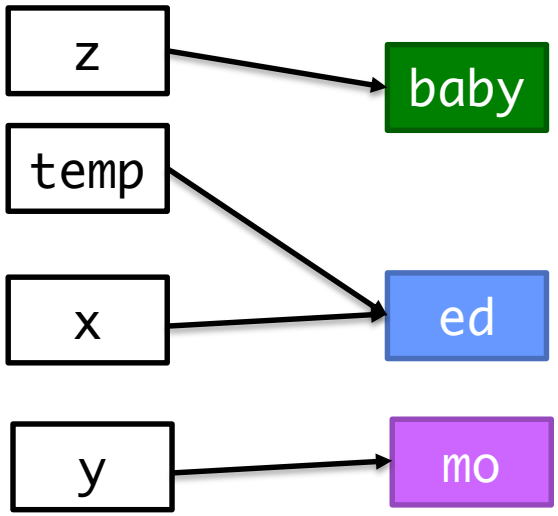
What Happens in This Code?

```
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;
```



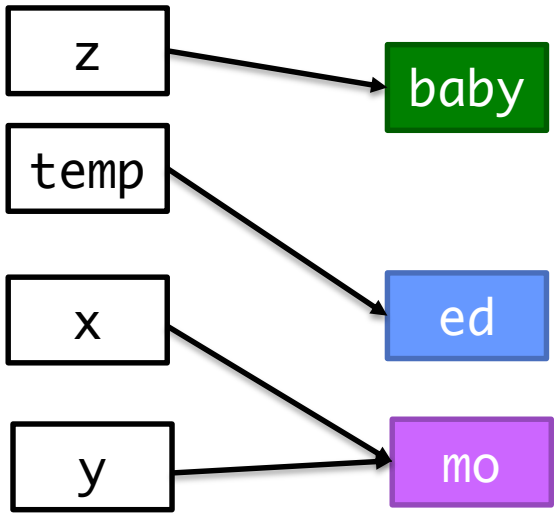
What Happens in This Code?

```
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;
```



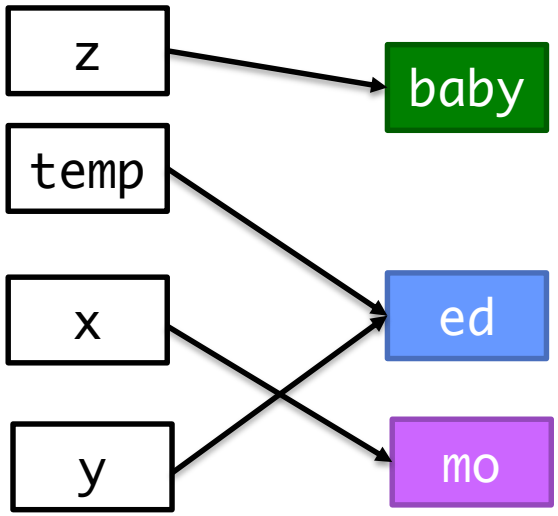
What Happens in This Code?

```
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;
```



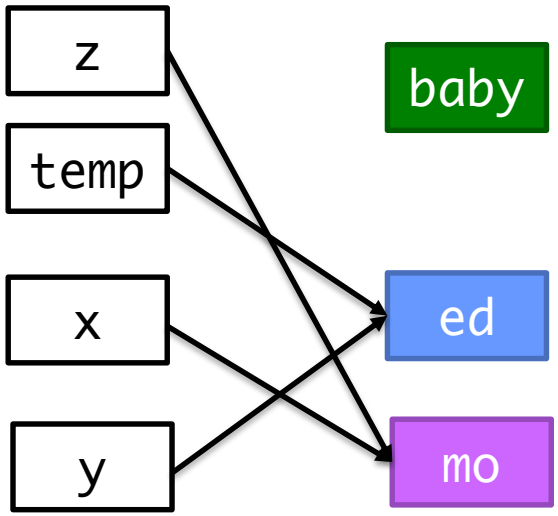
What Happens in This Code?

```
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;
```



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

GARBAGE COLLECTION

Memory Management

- Early languages (e.g., C): program needs to free memory when done with it
- In C++ and some other OOP languages, classes have explicit ***destructor*** methods that run when an object is no longer in scope
- Java provides ***automatic garbage collection***
  - Reclaims memory allocated for objects that are no longer referenced

Garbage Collector

- Garbage collector is low-priority thread
  - Or runs when available memory gets tight
  - i.e., it doesn't necessarily immediately free memory
- Before GC can clean up an object, the object may have opened resources
  - Ex: generated temp files or open network connections that should be deleted/closed first
- GC calls object's `finalize()` method
  - Object's chance to clean up resources

`finalize()`

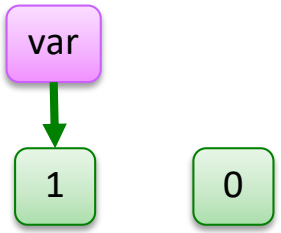
- Inherited from `java.lang.Object`
- Called before garbage collector sweeps away an object and reclaims its memory
- Should not be used for reclaiming resources
  - i.e., *close resources as soon as possible*
  - Why?
    - *When* method is called is not deterministic or consistent
    - Only know it will run sometime before garbage collection
- Clean up anything that cannot be atomically cleaned up by the garbage collector
  - Close file handles, network connections, database connections, etc.
- Note: no finalizer chaining
  - Must explicitly call parent object's `finalize` method

Alternatives to `finalize`

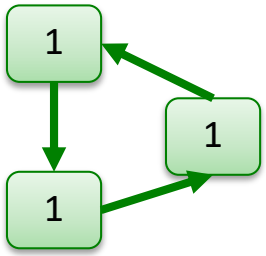
- Recall: unknown when `finalize` will execute—or *if* it will execute
  - Also *heavy performance cost*
- Solution: create your own terminating method
  - User of class terminates when done using object
- Examples: `File`'s or `Scanner`'s `close` method
- May still want `finalize()` as a safety net if user didn't call the terminate method
  - Log a warning message so user knows error in code

Do you know what Python does?

Python Garbage Collection



- Python also does garbage collection
- Python does ***reference counting***
  - On each reference/dereference, update the number of references to the object
    - Can't handle reference cycles



- Python also does ***generational garbage collection*** to handle reference cycles
- Tradeoffs with Java's Garbage Collection
  - Synchronous (not asynchronous) process (i.e., slows down execution)
  - Cheaper memory costs than Java for keeping track of what can be garbage collected

Discussion

- We discussed garbage collection (for both Java and Python)
- What are the benefits and limitations of garbage collection?
  - Compare with C, which does not have garbage collection

Garbage Collection

Benefits

- Programmer doesn't need to worry about memory management
- Cleans up unused memory automatically, eventually
- Programmer can never release memory that is then accessed (a.k.a. seg faults)

Drawbacks

- Programmer doesn't worry about memory management
  - May not be as careful to avoid memory leaks
- Memory could be cleaned up sooner
- Requires resources (CPU, memory) to keep track of memory
- Slows program execution

Garbage Collection

Benefits

- Programmer doesn't need to worry about memory management
- Cleans up unused memory automatically, eventually
- Programmer can never release memory that is then accessed

Drawbacks

- Programmer doesn't worry about memory management
  - May not be as careful to avoid memory leaks
- Memory could be cleaned up sooner
- Requires resources (CPU, memory) to keep track of memory
- Slows program execution

- Generally, programmer time is more valuable than computer resources.
- Generally, less buggy code is preferred to more efficient code.