Objectives
- Object-oriented programming in Java
  - Constructors
  - Default constructors
  - Static methods, variables
  - Inherited methods

Assign 1 Discussion
- Java Conventions:
  - Class names: begin with capital letter
  - Class constants: name with all capital letters, e.g., DIFFICULTY_SCORE
- Can fully-specify the class instead of importing class
  java.util.Arrays.sort(myArray);

Danger of a Large Library
- Lots of classes that seem like they’re what we want but aren’t

Review
- Why OO programming?
  - What are its components?
- What’s wrong with “white-box” programming?
  - How does Java help to enforce black-box programming?
- What is the syntax for defining a constructor?
- What is the syntax for defining a method?

Review: Objects
- How object does something doesn’t matter
- What object does matters (its functionality)
  - What object exposes to other objects
  - Referred to as “black-box programming” or encapsulation
  - Has public interface that others can use
  - Hides state from others

Object References
- Variable of type object: value is memory location

<table>
<thead>
<tr>
<th>Chicken</th>
<th>weight</th>
<th>height</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>2.0</td>
<td>38</td>
<td>Fred</td>
</tr>
<tr>
<td>two</td>
<td>3.0</td>
<td>45</td>
<td>Sallie Mae</td>
</tr>
</tbody>
</table>

An array (e.g., int[] array) is not an instance of a class, so we cannot call methods on it.
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

Multiple Object Variables

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

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

  sal = [ ]
  sal2 = [ ]

  Chicken

  weight = 3.0
  height = 45
  name = "Sallie Mae"

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

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

What happens here?

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

Whoops! Lost “baby” chicken!
Memory leak!
Luckily Java has garbage collectors to clean up the memory leak

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 method that has the same name must have different parameters
    - “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 Initialization

- If instance field is not explicitly set in constructor, automatically set to default value
  - Numbers set to zero
  - Booleans set to false
  - Object variables set to null
  - Local variables are not assigned defaults
- Do not rely on defaults
  - Code is harder to understand

Clean Code Recommendation:
Set all instance fields in the constructor(s)

Explicit Field Initialization

- If more than one constructor needs an instance field set to same value, the field can be set explicitly in the field declaration

```java
class Chicken {
    private String name = "";
    ...
}
```

Set value here for all constructors

Explicit Field Initialization

- Or in a static method call

```java
class Employee {
    private int id = assignID();
    ...
    private static int assignID() {
        ...
    }
}
```

More on static later...

Explicit Field Initialization

- Explicit field initialization happens before any constructor runs
- A constructor can change an instance field that was set explicitly
- If the constructor does not set the field explicitly, explicit field initialization is used

```java
class Chicken {
    private String name = "";
    public Chicken( String name, .. ) {
        this.name = name;
    }  
    ...
}
```

Change explicit field initialization

final keyword

- An instance field can be final
- final instance fields must be set in the constructor or in the field declaration
  - Cannot be changed after object is constructed

```java
private final String dbname = "invoices";
private final String id;
...
public MyObject( String id ) {
    this.id = id;
}
```
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

Constructors Calling Constructors

- Can call a constructor from inside another constructor
- The `this` 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?
- Reduce code size/reduce duplicate code
- Ex: if name not provided, use default name
  ```java
  Chicken(int height, double weight) {
    this("Bubba", height, weight);
  }
  ```
- Another example:
  ```java
  Chicken(int height, double weight) {
    this();
    this.height = height;
    this.weight = weight;
  }
  ```
  Not in example code online

Parent Class: Object

- Every new class you create automatically inherits from the `Object` class
  - See Java API
- Useful `Object` methods to customize your class
  - String `toString`
    - Returns a string representation of the object
      - Like Python's `__str__`
  - boolean `equals(Object o)`
    - Return true iff this object and `o` are equivalent
      - Like Python's `__eq__`
  - void `finalize()`
    - Called when object is destroyed
      - Clean up resources

More on `toString()`

- Automatically called when object is passed to print methods
- Default implementation: Class name followed by @ followed by unsigned hexadecimal representation of hashcode
  - Example: `Chicken@163b91`
- General contract:
  - "A concise but informative representation that is easy for a person to read"
- Your responsibility: Document the format
Chicken.java toString

• What would be a good string representation of a Chicken object?
  ➢ Look at output before and after toString method implemented

boolean equals(Object o)

• Procedure (Source: Effective Java)
  ➢ Use the == operator to check if the argument is a reference to this object
  ➢ Use the instanceof operator to check if the argument has the correct type
    ➢ If a variable is a null reference, then instanceof will be false
  ➢ Cast the argument to the correct type
  ➢ For each “significant” field in the class, check if that field of the argument matches the corresponding field of this object
    ➢ For doubles, use Double.compare and for floats use Float.compare

@Override

• Annotation
  ➢ Tells compiler “This method overrides a method in a parent class. It should have the same signature as that method in the parent class”
  ➢ If you do not correctly override the method, then the compiler will give you a warning
  ➢ The point: use @Override so you don’t make silly —yet costly—mistakes

Next Time

• Assignment 3 – Birthday additions
  ➢ The Birthday Paradox

How should we determine that two Chickens are equivalent?