Objectives

- Object-oriented programming in Java
  - Encapsulation
  - Access modifiers
  - Using other's classes
  - Defining own classes

Review

- What two basic classes did we discuss?
- What do the following control structures look like in Java?
  - If
  - While
  - For
- What is the syntax for logic operators in Java?
- How do you create an array?
  - How do you determine the size of an array?

Review: Object-Oriented Programming

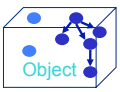
- Benefits?
- Components?

Review: Object-Oriented Programming

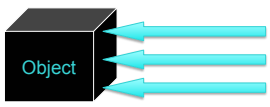
- Programming that models real life
  - Consider an ATM…
    - Implicitly agreed upon interface between user and the ATM
    - **What**, not how
  - Objects each have own role/responsibility
- As opposed to ***functional*** programming
  - A list of instructions to the computer

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**"



• Can see and manipulate object's internals



• Has public **interface** that others can use
• Hides state from others

Objects: Black-box programming

- If an object allows you to access and store data, you don't care if the underlying data type is an array or hashtable, etc.
  - Code just has to ***work***!
- Similarly, if object *sorts*, does not matter if uses merge or quick sort
- Problem with white-box:
  - What if implementation changes?
    - For scalability, efficiency, …

Access Modifiers

- A `public` method (or instance field) means that any object *of any class* can directly access the method (or field)
  - Least restrictive
- A `private` method (or instance field) means that any object *of the same class* can directly access this method (or field)
  - Most restrictive
- Additional access modifiers will be discussed with Inheritance

Classes & Objects

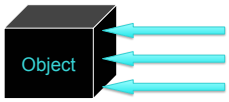
- Classes define template from which objects are made
  - "Cookie cutters"
  - Define **state -** data, usually `private`
  - Define **behavior** – an object's *methods*, usually `public`
    - Exceptions?
- Many objects can be created for a class
  - Object: the cookie!
  - Ex: Many Mustangs created from Ford's "blueprint"
  - Object is an **instance** of the class

Classes, Objects, Methods

- An object's state is stored in **instance fields**
- **Method**: sequence of instructions that access/modify an object's data
  - **Accessor**: accesses (doesn't modify) object
  - **Mutator**: changes object's data

Encapsulation

- **Encapsulation**: combining data and behavior (functionality) into one package (the object) and hiding the implementation of the data from the user of the object



Constructors

- **Constructor:** a special method that constructs and initializes an object
  - After construction, can call methods on object
- Constructors have the same name as their classes

Constructing objects using new

- Given the `File` **constructor**
  `File( String pathname)`
- Create a new `File` object using new keyword

```
File myFile = new File("debug.out");
```

Type/Classname

Effective Java: Code Inefficiency

- Avoid creating unnecessary objects:

```
String s = new String("text"); // DON'T DO THIS
```

- Do this instead:

```
String s = "text";
```

- Why?

Calling Methods

- Similar to Python

```
<objectname>.<methodname>(<parameters>);
```

- Saw examples with `String` class
- To call `static` methods, use

```
<classname>.<methodname>(<parameters>);
```


Using Other's Classes: Random

- Problem: write a Java program that prints "heads" or "tails" at random.
- Look at API of Random
  - What functionality is available?
  - How do you use the class?

CREATING YOUR OWN CLASSES

Classes and Objects

- Java is pure object-oriented programming
  - All data and methods in a program must be contained within a class
- But, for data, we have primitive types (e.g., `int, float, char`) as well as objects

Example: `Chicken` class

- State
  - Name, weight, height
- Behavior
  - Accessor methods
    - `getWeight, getHeight, getName`
    - Convention: "get" for "getter" methods
  - Mutator methods
    - `feed`: adds weight and height when bird eats
    - `changeName`

General Java Class Structure

```
public class ClassName {

   // --------- INSTANCE VARIABLES ---------------
   // define variables that represent object's state
   private int inst_var;

   // --------- CONSTRUCTORS ---------------
   public ClassName() {
      // initialize data structures
   }

   // ----------- METHODS ------------
   public int getInfo() {
      return inst_var;
   }
}
```

Note: instance variables are `private` and methods are `public`

Example: `Chicken` class

- State
  - Name, weight, height
- Behavior
  - Accessor methods
    - `getWeight, getHeight, getName`
    - Convention: "get" for "getter" methods
  - Mutator methods
    - `eat`: adds weight
    - changeName

Discussion: data types for state variables?

Instance Variables: `Chicken.java`

```
public class Chicken {

   // --------- INSTANCE VARIABLES ---------------
   private String name;
   private int height; // in cm
   private double weight; // in lbs
```

All instance variables are **`private`**

Constructor: `Chicken.java`

```
public class Chicken {

   // --------- INSTANCE VARIABLES ---------------
   private String name;
   private int height; // in cm

   // --------- CONSTRUCTORS ---------------
   public Chicken(String name, int height,
                                double weight) {
      this.name = name;
      this.height = height;
      this.weight = weight;
   }
   …
```

Constructor name same as class's name

Type and name for each parameter

`this`: Special name for the current object, like `self` in Python (differentiate from parameters)

Example: `Chicken` class

- State
  - Name, weight, height
- Behavior
  - Accessor methods
    - `getWeight, getHeight, getName`
    - Convention: "get" for "getter" methods
  - Mutator methods
    - `eat`: adds weight
    - changeName

Discussion: What are the methods' **input** (parameters) and **output** (what is returned)?

Methods: `Chicken.java`

```
   …   Type the method returns

// --------- Getter Methods ---------------
   public String getName() {
      return name;
   }

// --------- Mutator Methods ---------------
   public void feed() {
      weight += .2;
      height += 1;
   }
   …
}
```

Chicken object's instance variables

Note that you don't *have* to use **`this`** when variables are unambiguous

Constructing objects

- Given the `Chicken` **constructor**

```
Chicken( String name, int height, double
  weight )
```

create three chickens

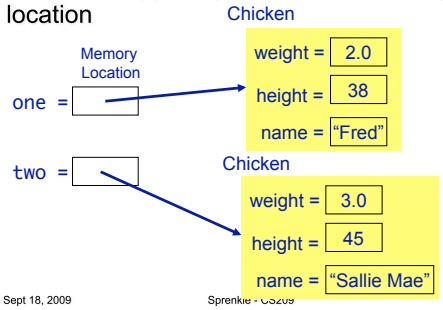
- "Fred", weight: 2.0, height: 38
- "Sallie Mae", weight: 3.0, height: 45
- "Momma", weight: 6.0, height: 83

Using Classes You Wrote

- In `Chicken.java`, call methods on the constructed objects

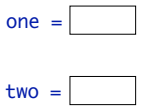
Object References

- Variable of type object: value is memory location



Object References

- Variable of type object: value is memory location



If I haven't called the constructor, only *declared* the variables:

```
Chicken one;
Chicken two;
```

Both `one` and `two` are equal to `null`

Null Object Variables

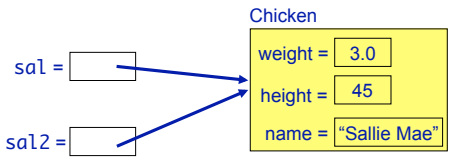
- An object variable can be explicitly set to `null`
  - Means that the object variable does not currently refer to any object
- It is possible to test if an object variable is set to `null`

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


Multiple Object Variables

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

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



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;
```


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

Assignment 2

- Part 1: Debugging
- Part 2: Writing a Birthday class (will build on later)
- Due Monday before class