

Objectives

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

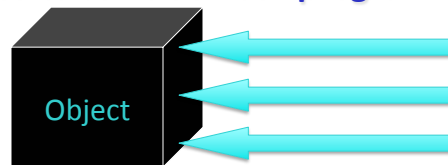
Review: Object-Oriented Programming

- What is OO programming?
 - Components?

- Benefits?

Review: Objects

- **How** object does something doesn't matter
 - Example: if object *sorts*, does not matter if uses merge or quick sort
- **What** object does matters (its **functionality**)
 - What object *exposes* to other objects
 - Referred to as “**black-box programming**”



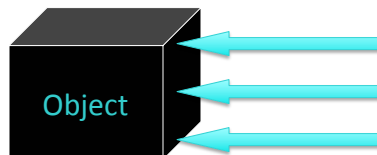
- Has public **interface** that others can use
- Hides state from others

Sept 16, 2016

3

Property: 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



- Java's characteristics allow us to enforce encapsulation better than Python

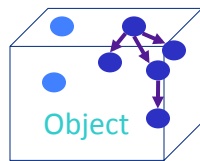
Sept 16, 2016

Sprenkle - CSCI209

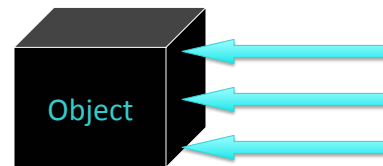
4

Discussion

- What is the problem with white-box programming?



Can see and manipulate object's internals



Sept 16, 2016

Sprenkle - CSCI209

5

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

Sept 16, 2016

Sprenkle - CSCI209

6

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

Sept 16, 2016

Sprenkle - CSCI209

7

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

In general, what access modifiers will we use for methods? For instance fields?

Sept 16, 2016

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

Sept 16, 2016

Sprenkle - CSCI209

9

Constructing objects using **new**

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

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

↑
Type/Classname

Sept 16, 2016

Sprenkle - CSCI209

10

Calling Methods

- Similar to Python

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

- Examples with `String` and `System` classes

- Review: to call `static` methods, use

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

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, can use objects as well as primitive types (e.g., **int**, **float**, **char**)

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
 - **setName**



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**

Sept 16, 2016

Sprenkle - CSCI209

15

Example: Chicken class

- State
 - Name, weight, height
- Behavior
 - Accessor methods
 - getWeight, getHeight, getName
 - Convention: “get” for “getter” methods
 - Mutator methods
 - feed: adds weight, height
 - setName

Discussion: data types for state variables?



Sept 16, 2016

Sprenkle - CSCI209

16

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
    private double weight;

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

Observations?

Constructor: Chicken.java

```
public class Chicken {
    // ----- INSTANCE VARIABLES -----
    private String name;
    private int height; // in cm
    // ----- CONSTRUCTORS -----
    public Chicken(String name, int h,
                   double weight) {
        this.name = name;
        this.height = h;
        this.weight = weight;
    }
    ...
}
```

Constructor name same as class's name

Type and name for each parameter

Params don't need to be same names as instance var names

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

Sept 16, 2016

Sprenkle - CSCI209

19

Example: Chicken class

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



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

Sept 16, 2016

Sprenkle - CSCI209

20

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

Sept 16, 2016

Sprenkle - CSCI209

21

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

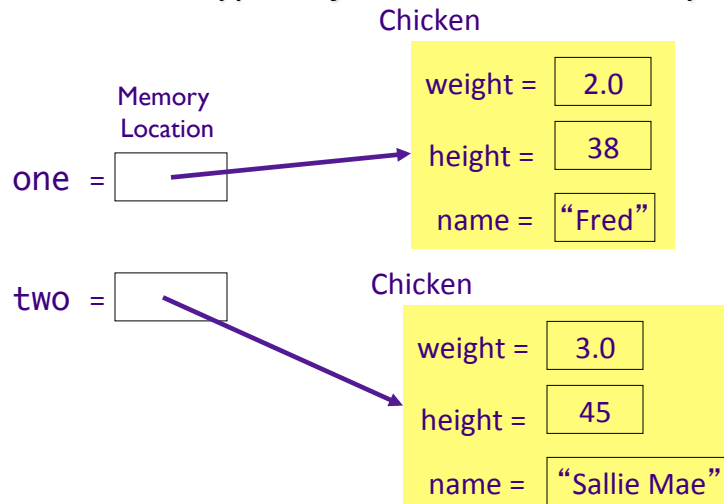
Sept 16, 2016

Sprenkle - CSCI209

22

Object References

- Variable of type object: value is memory location



Sept 16, 2016

Sprenkle - CSCI209

23

Object References

- Variable of type object: value is memory location

one =

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

two =

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

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

This is the case for *objects*. Primitive types are not `null`.

Sept 16, 2016

Sprenkle - CSCI209

24

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`

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

Sept 16, 2016

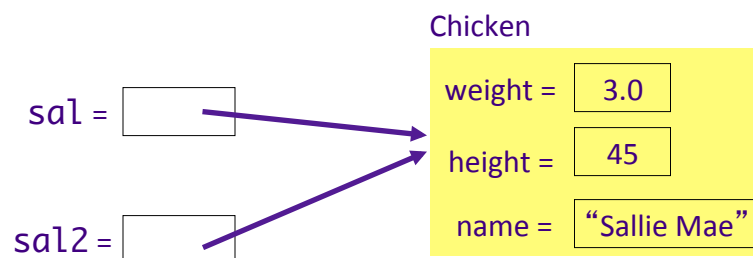
Sprenkle - CSCI209

25

Multiple Object Variables

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

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



Sept 16, 2016

Sprenkle - CSCI209

26

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

Sept 16, 2016

Sprenkle - CSCI209

27

Example of Overloaded Constructors

Constructors

Constructor and Description

File(File parent, String child)

Creates a new File instance from a parent abstract pathname and a child pathname string.

File(String pathname)

Creates a new File instance by converting the given pathname string into an abstract pathname.

File(String parent, String child)

Creates a new File instance from a parent pathname string and a child pathname string.

File(URI uri)

Creates a new File instance by converting the given file: URI into an abstract pathname.

Sept 16, 2016

Sprenkle - CSCI209

28

Constructor Overloading

- Allowing > 1 constructor (or any method) with the same name is called **overloading**
 - Constraint: Each of the methods that have the same name must have different parameters so that compiler can distinguish between them
 - “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

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

← Set value here for all constructors

Explicit Field Initialization


- Or in a static method call

```
class Employee {
    private int id = assignID();
    . . .
    private static int assignID() {
        int r = nextID;
        nextID++;
        return r;
    }
}
```

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

```
class Chicken {  
    private String name = "";  
    public Chicken( String name, ... ) {  
        this.name = name;  Change explicit  
        ... field initialization  
    }  
    ...  
}
```

33