

## Objectives

- Object-oriented programming in Java

Sept 12, 2008

Sprenkle - CS209

1

## Review

- What do the following control structures look like in Java?
  - If
  - While
  - For

Sept 12, 2008

Sprenkle - CS209

2

## Control Flow: foreach Loop

- Introduced in Java 1.5
  - Sun calls "enhanced for" loop
- Iterate over all elements in an array (or Collection)
  - Similar to Python's for loop

```
int[] a;
int result = 0;
for (int i : a) {
    result += i;
}
```

for each int element i in the array a  
The loop body is visited once for each element of a.

Sept 12, 2008

Sprenkle - CS209

3

## Review: Object-Oriented Programming

- Benefits?
- Components?

Sept 12, 2008

Sprenkle - CS209

4

## Review: Object-Oriented Programming

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

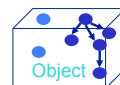
Sept 12, 2008

Sprenkle - CS209

5

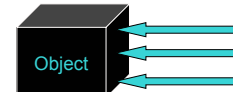
## Objects

- *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

Sept 12, 2008



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

Sprenkle - CS209

6

## 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, ...

Sept 12, 2008

Sprenkle - CS209

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
- Other access modifiers will be discussed with Inheritance

Sept 12, 2008

Sprenkle - CS209

8

## Classes & Objects

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

Sept 12, 2008

Sprenkle - CS209

9

## Classes, Objects, Methods

- Classes define template from which objects are made.
  - **State** - data, usually **private**
  - **Behavior** - *methods* for an object, usually **public**
- 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

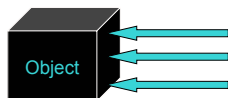
Sept 12, 2008

Sprenkle - CS209

10

## Encapsulation

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



Sept 12, 2008

Sprenkle - CS209

11

## Constructors

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

Sept 12, 2008

Sprenkle - CS209

12

## 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

Sept 12, 2008

Sprenkle - CS209

13

## Calling Methods

- Similar to Python  
`<objectname>.<methodname>(<parameters>);`
- Saw examples with String class
- To call **static** methods, use  
`<classname>.<methodname>(<parameters>);`

Sept 12, 2008

Sprenkle - CS209

14

## 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?

Sept 12, 2008

Sprenkle - CS209

CoinFlip.java 15

## CREATING YOUR OWN CLASSES

Sept 12, 2008

Sprenkle - CS209

16

## 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

Sept 12, 2008

Sprenkle - CS209

17

## Example: Chicken class

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



Sept 12, 2008

Sprenkle - CS209

18

## 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 12, 2008

Sprenkle - CS209

19

## 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: types for state variables?

Sept 12, 2008

Sprenkle - CS209

20

## Instance Variables: Chicken.java

```
public class Chicken {
    // ----- INSTANCE VARIABLES -----
    private String name;
    private int height; // in cm
    private double weight;
}
```

All instance variables are **private**

Sept 12, 2008

Sprenkle - CS209

21

## 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)

Sept 12, 2008

Sprenkle - CS209

22

## 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)?

Sept 12, 2008

Sprenkle - CS209

23

## Methods: Chicken.java

```
... Type the method returns

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

// ----- Mutator Methods -----
public void feed() {
    weight += .3;
    height += 1;
}
...
```

Chicken object's instance variables

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

Sept 12, 2008

Sprenkle - CS209

24

## 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 12, 2008

Sprenkle - CS209

25

## Using Classes You Wrote

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

Sept 12, 2008

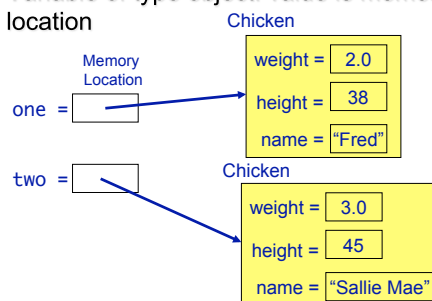
Sprenkle - CS209

`Chicken.java`

26

## Object References

- Variable of type object: value is memory location



Sept 12, 2008

Sprenkle - CS209

27

## 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`

Sept 12, 2008

Sprenkle - CS209

28

## 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) {
    . . .
}
```

Sept 12, 2008

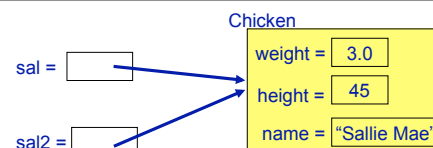
Sprenkle - CS209

29

## Multiple Object Variables

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

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



Sept 12, 2008

Sprenkle - CS209

30

## 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;
```

Sept 12, 2008

Sprenkle - CS209

31

## 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 take care of the memory leak

Sept 12, 2008

Sprenkle - CS209

32

## 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 12, 2008

Sprenkle - CS209

33

## 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
    - "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 wasn't that possible?

Sept 12, 2008

Sprenkle - CS209

34

## Default Initialization

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

Sept 12, 2008

Sprenkle - CS209

35

## 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 = "";
    . . .
}
```

Sept 12, 2008

Sprenkle - CS209

36

## 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...

Sept 12, 2008

Sprenkle - CS209

37

## 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

Sept 12, 2008

Sprenkle - CS209

38

## Assignment 2

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

Sept 12, 2008

Sprenkle - CS209

39