

Objectives

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

Sept 16, 2011

Sprenkle - CSCI209

1

Review

- What is the keyword for a constant value?
- What does **static** mean?
- What two Java 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?
- How can you sort an array?

Sept 16, 2011

Sprenkle - CSCI209

2

Assign0 Feedback

- Terminology clarification
 - Declaration: **int** x = 3;
 - Definition: x = 3;
- Comment for author: @author Dr. Seuss
 - Will make more sense when we talk more about JavaDocs
- 2 votes for Katy Perry

Sept 16, 2011

Sprenkle - CSCI209

3

What does this code do?

```
if ( x > 4 ) ;
    System.out.println("x is " + x);
```

Sept 16, 2011

Sprenkle - CSCI209

4

What does this code do?

```
if ( x > 4 ) ;
    System.out.println("x is " + x);
```

- ; is a valid statement
- Print statement *a*lways executes
- Indentation doesn't matter

Sept 16, 2011

Sprenkle - CSCI209

5

Control Flow: foreach Loop

- Introduced in Java 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*.

<http://download.oracle.com/javase/1.5.0/docs/guide/language/foreach.html> `ArrayLength.java`

Review: Object-Oriented Programming

- What is OO programming?
 - Components?
- Benefits?

Sept 16, 2011

Sprenkle - CSCI209

7

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

Sept 16, 2011

Sprenkle - CSCI209

8

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**"



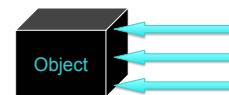
- Has public **interface** that others can use
- Hides state from others

Sept 16, 2011

9

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



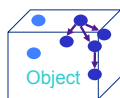
Sept 16, 2011

Sprenkle - CSCI209

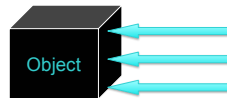
10

Discussion

- What is the problem with white-box programming?
- What if implementation changes?
 - For scalability, efficiency, ...



Can see and manipulate
object's internals



Sept 16, 2011

Sprenkle - CSCI209

11

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, 2011

Sprenkle - CSCI209

12

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, 2011

Sprenkle - CSCI209

13

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, 2011

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, 2011

Sprenkle - CSCI209

15

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, 2011

Sprenkle - CSCI209

16

Effective Java: Code Inefficiency

- Avoid creating unnecessary objects:

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

- Do this instead:

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

Why?

Sept 16, 2011

Sprenkle - CSCI209

17

Calling Methods

- Similar to Python

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

- Examples with String and System classes

- To call **static** methods, use

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

Sept 16, 2011

Sprenkle - CSCI209

18

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 16, 2011

Sprenkle - CSCI209 CoinFlip.java 19

CREATING YOUR OWN CLASSES

Sept 16, 2011

Sprenkle - CSCI209

20

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**)

Sept 16, 2011

Sprenkle - CSCI209

21

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



Sept 16, 2011

Sprenkle - CSCI209

22

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, 2011

Sprenkle - CSCI209

23

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, 2011

Sprenkle - CSCI209

24

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**

Sept 16, 2011

Sprenkle - CSCI209

25

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 constructed object, like SELF in Python (differentiate from parameters)

Sept 16, 2011

Sprenkle - CSCI209

26

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, 2011

Sprenkle - CSCI209

27

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, 2011

Sprenkle - CSCI209

28

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, 2011

Sprenkle - CSCI209

29

Using Classes You Wrote

- In **Chicken.java**,
 - Construct chickens
 - Call methods on the constructed objects

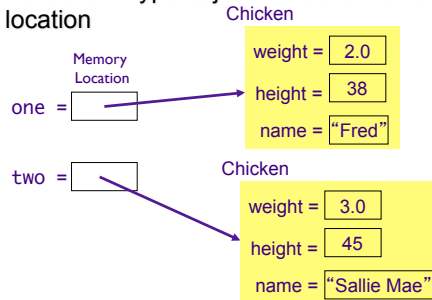
Sept 16, 2011

Sprenkle - CSCI209

Chicken.java 30

Object References

- Variable of type object: value is memory location



Sept 16, 2011

Sprenkle - CSCI209

31

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 16, 2011

Sprenkle - CSCI209

32

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, 2011

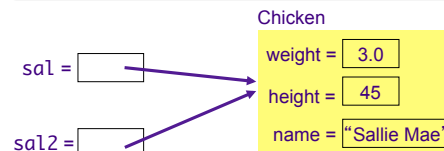
Sprenkle - CSCI209

33

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, 2011

Sprenkle - CSCI209

34

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 16, 2011

Sprenkle - CSCI209

35

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

Sept 16, 2011

Sprenkle - CSCI209

36

TODO

- Assignment 2:
 - Part 1: Debugging
 - Part 2: Writing a Birthday class (will build on later)
 - Due Monday before class
- Extra Credit Opportunity: Monday
 - Sylvia Earle: "The World is BLUE!"

Sept 16, 2011

Sprenkle - CSCI209

37

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, 2011

Sprenkle - CSCI209

38

Example of Overloaded Constructors

Constructor Summary

```
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.
```

Also saw an example in java.util.Random

Sept 16, 2011

Sprenkle - CSCI209

39

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?

Sept 16, 2011

Sprenkle - CSCI209

overload.py

40

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)

Sept 16, 2011

Sprenkle - CSCI209

41

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

Sept 16, 2011

Sprenkle - CSCI209

42

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 16, 2011

Sprenkle - CSCI209

43

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

44

Create Another Constructor for Chicken class

- All Chickens start small, only have a name

Sept 16, 2011

Sprenkle - CSCI209

45

Benefits of Static Typing

- Look at `dynamic_typing.py`
- Discussion questions
 - What is the type of `data` at the end of the program?
 - How difficult is this program to understand?
 - If you had to debug this program, how easy/difficult would it be?
 - What is a benefit of dynamic typing?

`alternative_dynamic_typing.py`

Sept 16, 2011

Sprenkle - CSCI209

46

Benefits of Static Typing

- Easier to remember type of variable
 - Know operations that can be executed on a variable of a certain type
- Compiler can check that you're only using valid operations for this type
- More benefits later this semester

Sept 16, 2011

Sprenkle - CSCI209

47

More Why Java?

- More **structure** emphasizes/requires better **design**

Sept 16, 2011

Sprenkle - CSCI209

48

Assign 1: java.util.Arrays

- `Arrays` is a class in `java.util`
- Methods for sorting, searching, `deepEquals`, fill arrays
- To use class, need `import` statement
 - Goes at top of program, before class definition

```
import java.util.Arrays;
```

`ArraysExample.java`

Sept 14, 2011

Sprenkle - CSCI209

49