

Objectives

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

Sept 15, 2008

Sprenkle - CS209

1

Gymnastics Score

- Minor efficiency issues with computing the score
 - Don't need to make copy of array (waste of time and space)

```
Arrays.sort(execution_scores);
double total = 0;
for( int i=1; i < execution_scores.length - 1; i++ ) {
    total += execution_scores[i];
}
double exec_avg = total/(execution_scores.length - 2);
```

No unnecessary additions
More efficient than doing check on i in loop

Sept 15, 2008

Sprenkle - CS209

2

Review

- What's wrong with "white-box" programming?
- What is the syntax for declaring a method?

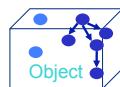
Sept 15, 2008

Sprenkle - CS209

3

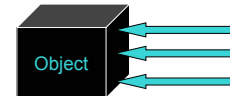
Review: Objects

- 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**"



- Can see and manipulate object's internals

Sept 15, 2008



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

Sprenkle - CS209

4

Review: 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 15, 2008

Sprenkle - CS209

5

final keyword

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

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

Sept 12, 2008

Sprenkle - CS209

6

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, the default constructor is NOT provided

Sept 15, 2008

Sprenkle - CS209

7

Default Constructor

- Chicken class has one constructor:


```
Chicken(String name, float weight, float height)
```
- No default constructor


```
Chicken chicken = new Chicken();
```

 - Is a compiler error

Sept 15, 2008

Sprenkle - CS209

8

Constructors Calling Constructors

- Can call a constructor from inside another constructor
- The first statement of constructor must be


```
this( . . . );
```

 to call another constructor of the same class.
 - this refers to the object being constructed

Sept 15, 2008

Sprenkle - CS209

9

Constructors Calling Constructors

- Why would you call another constructor?
 - Reduce code size/reduce duplicate code
- Ex: if name not provided, use default name


```
Chicken( float height, float weight ) {
    this( "Bubba", height, weight);
}
```
- Example: base case constructor


```
Chicken( float height, float weight ) {
    this();
    this.height = height;
    this.weight = weight;
}
```

Sept 15, 2008

Sprenkle - CS209

10

Parent Class: Object

- Every new class you create automatically inherits from the `Object` class
 - See Java API
- Useful 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__` or `__cmp__`
 - `void finalize()`
 - Called when object is destroyed
 - Clean up resources

Type signature

Sept 15, 2008

Sprenkle - CS209

11

Examples: Chicken.java

- What would be a good String representation of a Chicken object?
- How would we know if two Chickens are equal?

Sept 15, 2008

Sprenkle - CS209

12

Object Destructors

- In C++ (and many other OOP languages), classes have explicit destructor methods that run when an object is no longer used.
- Java does not support destructors, as it provides *automatic garbage collection*
 - Watches/waits until there are no references to an object
 - Reclaims the memory allocated for the object that is no longer used

Sept 15, 2008

Sprenkle - CS209

13

finalize()

- Called before garbage collector sweeps away the object and reclaims the memory
- This method should not be used for reclaiming any resources
 - i.e. close resources as soon as possible
 - Timing when this method is called is not deterministic or consistent
 - Only know it will run sometime before garbage collection

Sept 15, 2008

Sprenkle - CS209

14

Using finalize()

- Clean up anything that cannot be atomically cleaned up by the garbage collector
 - Close file handles
 - Close network connections
 - Close database connections
 - etc...

Sept 15, 2008

Sprenkle - CS209

15

Static Methods/Fields

- For related functionality/data that isn't specific to any particular object
- `java.lang.Math`
 - No constructor (what does that mean?)
 - Static fields: `PI`, `E`
 - Static methods:
 - `static double sin(double a)`

Sept 15, 2008

Sprenkle - CS209

16

Static Methods

- Do not operate on objects
- Cannot access instance fields of their class
- Can access *static fields* of their class
- Sort of like Python *functions* that are associated with the class

Sept 15, 2008

Sprenkle - CS209

17

Static Fields

- A static field is used when only one such field **per class** (not object!)
- All objects of a class share **one copy** of the static field

Sept 15, 2008

Sprenkle - CS209

18

Static Fields Example

```
public class Student {
    private int id;
    private static int nextID = 1;
    ...
}
```

- Each **Student** object has an **id** field, but there is only one **nextID** field, shared among all instances of the class
 - **nextID** field exists even when no **Student** objects have been constructed

How would we use the **nextID** field to create unique IDs?

Sept 15, 2008

Sprenkle - CS209

19

Static Field Example

- One option:

```
public class Student {
    private int id = assignID();
    private static int nextID = 1;
    private static int assignID() {
        int r = nextID;
        nextID++;
        return r;
    }
    ...
}
```

Sept 15, 2008

Sprenkle - CS209

20

Constant Static Fields

- We also used a static field to designate a constant in a class...


```
public class Converter{
    ...
    public static final
    double CM2IN= 2.54;
}
```
- The **Math** class has a static constant, **PI**
 - The value can be accessed using the **Math** class: `a = b * Math.PI;`
- Notice we do not need to create an object of the **Math** class to use this constant
 - What is another benefit of a class constant?

Sept 15, 2008

Sprenkle - CS209

21

main()

- Most popular static method we have seen
- main()** does not operate on any objects
 - Runs when a program starts...there are no objects yet
- main()** executes and constructs the objects the program needs and will use
 - Like the *driver function* for the program

Sept 15, 2008

Sprenkle - CS209

22

Analyzing java.lang.String

- String toUpperCase()**
 - Converts all of the characters in *this* **String** to upper case
- static String valueOf(boolean b)**
 - Returns the string representation of the **boolean** argument
- Discussion: Why can the second method be **static**?

Sept 15, 2008

Sprenkle - CS209

23

Static Summary

- Static fields and methods are part of a class and not an object
 - Do not require an object of their class to be created in order to use them
- When would we make a method **static**?
 - When a method does not have to access an object's state (fields) because all needed data are passed into the method
 - When a method only needs to access static fields in the class

Sept 15, 2008

Sprenkle - CS209

24

Review: Class Design/Organization

- Fields
 - Chosen first
 - Placed at the beginning or end of the class
 - Has an accessor modifier, data type, variable name and some optional other modifiers
 - If no accessor modifier, defaults to **public**
 - Use **this** keyword to access the object
- Constructors
- Methods
 - Need to declare the return type
 - May be **public static ...**

Sept 15, 2008

Sprenkle - CS209

25

Encapsulation Revisited

- Objects should hide their data and only allow other objects to access this data through **accessor** and **mutator** methods.
- Common programmer mistake:
 - Creating an accessor method that returns a reference to a mutable (changeable) object.

Sept 15, 2008

Sprenkle - CS209

26

What is "bad" about this class?

```
class Farm {
    . . .
    private Chicken headRooster;

    public Chicken getHeadRooster() {
        return headRooster;
    }
    . . .
}
```

Sept 15, 2008

Sprenkle - CS209

27

Fixing the Problem: Cloning

```
class Farm {
    . . .
    private Chicken headRooster;

    public Chicken getHeadRooster() {
        return (Chicken) headRooster.clone();
    }
    . . .
}
```

Method is available to all objects
(inherited from Object)

- In previous example, could modify returned object's state
- Another **Chicken** object, with the same data as **headRooster**, is created and returned to the user.
- If the user modifies (e.g., feeds) that object, **headRooster** is not affected

Sept 15, 2008

Sprenkle - CS209

28

Cloning

- Cloning is a more complicated topic than it seems from the example.
- We may examine cloning in more detail later

Sept 15, 2008

Sprenkle - CS209

29

Assignment 3

- Static method practice
- Modifying the **Birthday** class
 - **toString**, **equals**, another constructor
- Using the **Birthday** class to show probability of two people having same birthday

Sept 15, 2008

Sprenkle - CS209

30