

Objectives

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

Sept 19, 2011

Sprenkle - CSCI209

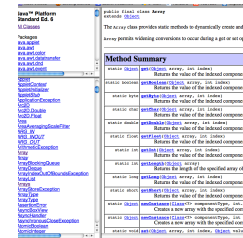
1

Danger of a Large Library

- Lots of classes that seem like they're what we want but aren't

`java.lang.reflect.Array`
`javax.sql.rowset.serial.Array`

An array (e.g., `int[]` array) is **not** an instance of a class, so we cannot call methods on it.



Sept 19, 2011

Sprenkle - CSCI209

2

Assign 1 Discussion

- Conventions:
 - Class names: begin with capital letter
 - Class constants: name with all capital letters, e.g., `DIFFICULTY_SCORE`
- Identifying inefficient code:

```
public static void main(String[] args) {
    // handle command-line args
    // ...

    String aString = args[0];
    StringBuffer str = new StringBuffer(args[0]);
    System.out.println(str + " backwards is " +
        str.reverse());
}
```

Sept

Review

- Why OO programming?
 - What are its components?
- What's wrong with "white-box" programming?
- What is the syntax for defining a constructor?
- What is the syntax for defining a method?

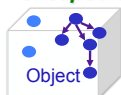
Sept 19, 2011

Sprenkle - CSCI209

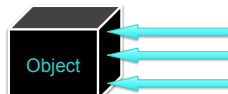
4

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**" or **encapsulation**



- Can see and manipulate object's internals



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

Sept 19, 2011

Sprenkle - CSCI209

5

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 19, 2011

Sprenkle - CSCI209

6

Constructor Overloading

- Allowing > 1 constructor (or any method) with the same name is called **overloading**
 - Constraint: Each method that has 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 isn't overloading possible in Python?

Sept 19, 2011

Sprenkle - CSCI209

overload.py

7

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 19, 2011

Sprenkle - CSCI209

8

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 19, 2011

Sprenkle - CSCI209

9

Explicit Field Initialization

- Or in a static method call

```
class Employee {
    private int id = assignID();
    ...
    private static int assignID() {
        ...
    }
}
```

More on static later...

Sept 19, 2011

Sprenkle - CSCI209

10

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

11

final keyword

- An instance field can be **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 19, 2011

Sprenkle - CSCI209

12

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
 - Default constructor is **NOT** provided

Sept 19, 2011

Sprenkle - CSCI209

13

Default Constructor

- Chicken class has one constructor:


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


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

 - Is a compiler error

Sept 19, 2011

Sprenkle - CSCI209

14

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

Why would you want to call another constructor?

Sept 19, 2011

Sprenkle - CSCI209

15

Constructors Calling Constructors

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


```
Chicken( int height, double weight ) {
    this( "Bubba", height, weight);
}
```
- Another example

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

Not in example code online

Sept 19, 2011

Sprenkle - CSCI209

16

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

Method signature

Sept 19, 2011

Sprenkle - CSCI209

17

More on toString()

- Automatically called when object is passed to print methods
- Default implementation: Class name followed by `@` followed by unsigned hexadecimal representation of hashcode
 - Example: `Chicken@163b91`
- General contract: "A concise but informative representation that is easy for a person to read"
- Your responsibility: Document the format

Sept 19, 2011

Sprenkle - CSCI209

18

Examples: Chicken.java

- What would be a good string representation of a Chicken object?
 - Look at output before and after toString method implemented
- How would we know if two Chickens are equivalent?

Sept 19, 2011

Sprenkle - CSCI209

19



GARBAGE COLLECTION

Sept 19, 2011

Sprenkle - CSCI209

20

Memory Management

- In C++ and some other OOP languages, classes have explicit destructor methods that run when an object is no longer used
- Java does not support destructors because it provides **automatic garbage collection**
 - Waits until there are no references to an object
 - Reclaims memory allocated for the object that is no longer referenced

Do you know what Python does?

Sept 19, 2011

Sprenkle - CSCI209

21

Garbage Collector

- Garbage collector is low-priority thread
 - Or runs when available memory gets tight
- Before GC can clean up an object, the object may have opened resources
 - Ex: generated temp files or open network connections that should be deleted/closed first
- GC calls object's finalize() method
 - Object's chance to clean up resources

Discussion: Benefits and costs of garbage collection?

Sept 19, 2011

Sprenkle - CSCI209

22

Garbage Collection

Benefits

- Fewer memory leaks
 - Less buggy code
 - But, memory leaks are still possible
- Code is easier to write

Costs

- Garbage collection may not be as efficient as explicit freeing memory



Sept 19, 2011

Sprenkle -

finalize()

- Inherited from java.lang.Object
- Called before garbage collector sweeps away an object and reclaims its memory
- Should not be used for reclaiming resources
 - i.e., close resources as soon as possible
 - Why?
 - When method is called is not deterministic or consistent
 - Only know it will run sometime before garbage collection
- Clean up anything that cannot be atomically cleaned up by the garbage collector
 - Close file handles, network connections, database connections, etc.
- Note: no finalizer chaining
 - Must explicitly call parent object's finalize method

Sept 19, 2011

Sprenkle - CSCI209

24

Alternatives to finalize

- Recall: unknown when `finalize` will execute—or *if* it will execute
 - Also *heavy performance cost*
- Solution: create your own terminating method
 - User of class terminates when done using object
- Examples: File's or Window's close method
- May still want a `finalize` method as a safety net if user didn't call the terminate method
 - Log a warning message so user knows error in code

Sept 19, 2011

Sprenkle - CSCI209

25

STATIC METHODS AND FIELDS

Sept 19, 2011

Sprenkle - CSCI209

26

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 19, 2011

Sprenkle - CSCI209

27

Static Methods

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

Sept 19, 2011

Sprenkle - CSCI209

28

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 19, 2011

Sprenkle - CSCI209

29

Constant Static Fields

- We used a static field to designate a *class constant*:

```
public class Converter {
    public static final double CM2IN = 2.54;
```

- The `Math` class has a static constant, `PI`
 - Value can be accessed using the `Math` class:


```
area = Math.PI * r * r;
```
- Do not need an object of the `Math` class to use this constant

Sept 19, 2011

Sprenkle - CSCI209

30

Static Fields Example

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

- 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 could we use the **nextID** field to create unique IDs?

Sept 19, 2011

Sprenkle - CSCI209

31

Static Field Example

```
public class Student {
    private static int nextID = 1;
    private int id = assignID();

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

Sept 19, 2011

Sprenkle - CSCI209

32

main()

- Most common static method
- 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 19, 2011

Sprenkle - CSCI209

33

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

Why can the second method be **static**?

Sept 19, 2011

Sprenkle - CSCI209

34

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 19, 2011

Sprenkle - CSCI209

35

Review: Class Design/Organization

- Fields**
 - Chosen first
 - Placed at the beginning or end of class definition
 - Have an access modifier, data type, variable name, and some optional other modifiers
 - If no access modifier, defaults to **package-private**
 - Use **this** keyword to access the object
- Constructors**
- Methods**
 - Need to declare the return type
 - Have an access modifier (defaults to **package-private** if none specified)

Sept 19, 2011

Sprenkle - CSCI209

36

TO DO

- Assignment 3
 - Static method practice
 - Modifying the Birthday class
 - toString, equals
 - Using Birthday class to show probability of two people having same birthday
- Extra Credit:
 - Answer questions on Sakai about Sylvia Earle's talk