

Objectives

- Garbage Collection
- Packages
- More on Inheritance
- Exam questions

Oct 5, 2022

Sprenkle - CSCI209

1

1

Review

- What are the benefits of programmatically testing (i.e., having the program determine if the test case fails)?
- Some code is returning a private variable from a public method
 - Why could that be a problem?
 - How should we implement that method?
- How does Java pass parameters?
 - What does that mean?
 - What are the consequences of that choice? (How does that affect how we call methods?)
- Review from CSCI-112:
 - What are the benefits of inheritance?
 - What are examples of inheritance?
 - When should you use inheritance?

Oct 5, 2022

Sprenkle - CSCI209

2

2

Review: Providing Private Data

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

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

Method is available to all objects
(inherited from Object)

- 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

Oct 5, 2022

Sprenkle - CSCI209

3

3

Review: Method Parameters in Java

- Java always passes parameters into methods **by value**
 - Meaning: the formal parameter becomes a copy of the argument/actual parameter's value
 - Method caller and callee have two independent variables with the same value
 - Consequence: Methods **cannot** change the **variables** used as input parameters

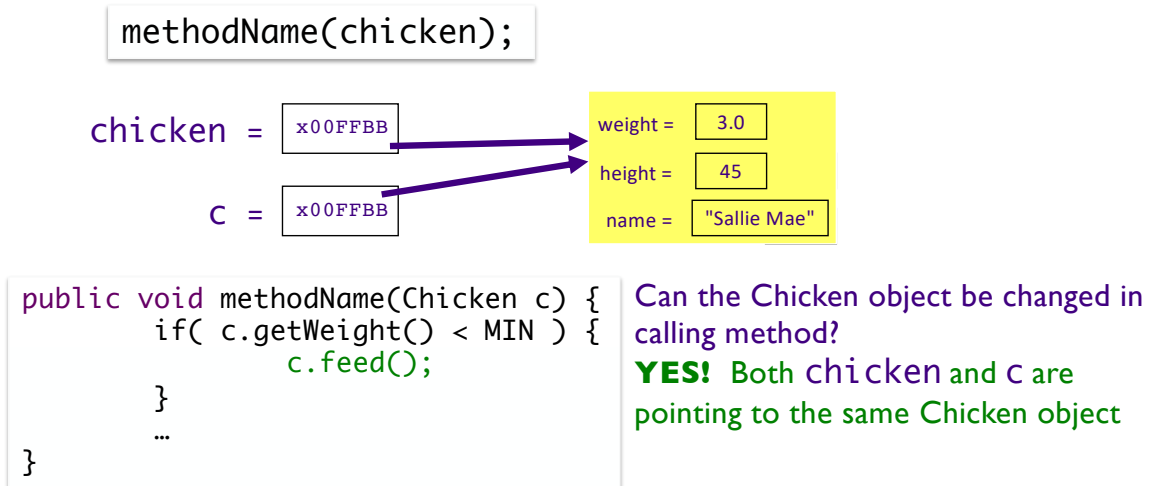
Oct 5, 2022

Sprenkle - CSCI209

4

4

Review: Pass by Value: Objects



Oct 5, 2022

Sprenkle - CSCI209

5

5

Review: Summary of Method Parameters

- Everything is passed **by value** in Java
 - Formal parameter copies the actual parameter
- An **object variable** (not an object) is passed into a method
 - Changing the *state* of an object in a method changes the state of object outside the method
 - Method does not see a copy of the original object

Oct 5, 2022

Sprenkle - CSCI209

6

6

What Happens in This Code?

```
Chicken x, y;  
Chicken z = new Chicken("baby", 5, 1.0);  
x = new Chicken("ed", 81, 10.3);  
y = new Chicken("mo", 63, 6.2);  
Chicken temp = x;  
x = y;  
y = temp;  
z = x;
```

baby

Whoops! Lost "baby" chicken! -- No object variable references it
Memory leak!

Luckily Java has *garbage collectors* to clean up the memory leak

Oct 5, 2022

Sprenkle - CSCI209

7

7

GARBAGE COLLECTION

Oct 5, 2022

Sprenkle - CSCI209

8

8

Memory Management

- Early languages (e.g., C): free memory when you're done with it
- In C++ and some other OOP languages, classes have explicit **destructor** methods that run when an object is no longer in scope
- Java provides **automatic garbage collection**
 - Reclaims memory allocated for objects that are no longer referenced

Oct 5, 2022

Sprenkle - CSCI209

9

9

Garbage Collector

- Garbage collector is low-priority thread
 - Or runs when available memory gets tight
 - i.e., it doesn't necessarily immediately free memory
- 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

Oct 5, 2022

Sprenkle - CSCI209

10

10

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

Oct 5, 2022

Sprenkle - CSCI209

11

11

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 `Scanner`'s `close` method
- May still want `finalize()` as a safety net if user didn't call the terminate method
 - Log a warning message so user knows error in code

Oct 5, 2022

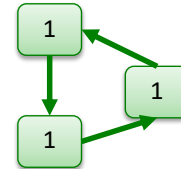
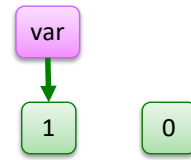
Do you know what Python does?

12

12

Python Garbage Collection

- Python also does garbage collection
- Python does **reference counting**
 - On each reference/dereference, update the number of references to the object
 - Can't handle reference cycles
- Python also does **generational garbage collection** to handle reference cycles
- Tradeoffs with Java's Garbage Collection
 - Synchronous (not asynchronous) process (i.e., slows down execution)
 - Cheaper memory costs than Java for keeping track of what can be garbage collected



Oct 5, 2022

Discussion: Benefits and limitations of garbage collection?

13

13

Garbage Collection

Benefits

- Programmer doesn't need to worry about memory management
- Cleans up unused memory automatically, eventually
- Programmer can never release memory that is then accessed (a.k.a. seg faults)

Drawbacks

- Programmer doesn't worry about memory management
 - May not be as careful to avoid memory leaks
- Memory could be cleaned up sooner
- Requires resources (CPU, memory) to keep track of memory
- Slows program execution

Oct 5, 2022



Sprenkle - CSCI209

14

14

Garbage Collection

Benefits

-  Programmer doesn't need to worry about memory management
-  Cleans up unused memory automatically, eventually
 - Programmer can never release memory that is then accessed
 - Generally, programmer time is more valuable than computer resources.
 - Generally, less buggy code is preferred to more efficient code.

Drawbacks

- Programmer doesn't worry about memory management
 - May not be as careful to avoid memory leaks
- Memory could be cleaned up sooner
- Requires resources (CPU, memory) to keep track of memory
- Slows program execution

Sprenkle - CSCI209

15

15

PACKAGES

Oct 5, 2022

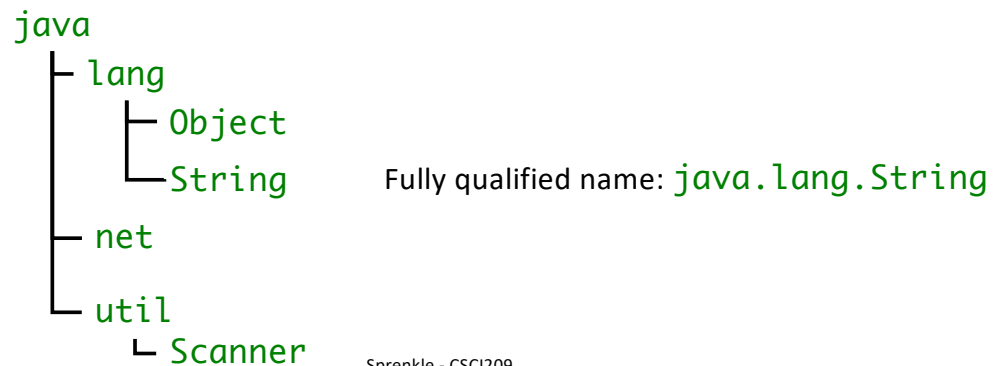
Sprenkle - CSCI209

16

16

Review: Packages

- Hierarchical structure of Java classes
 - Similar to Python's *modules*
 - Directories of directories



Oct 5, 2022

Sprenkle - CSCI209

17

17

Importing Packages

- Can import one class at a time or all the classes within a package
- Examples:

```
import java.util.Date;
import java.util.*; ← Import entire java.util package
```

- *** form may increase compile time**
 - BUT, no effect on run-time performance

Oct 5, 2022

Sprenkle - CSCI209

18

18

Why Packages?

- Organizes code
- Reduces chance of a conflict between names of classes
 - Example: Java's library has two classes named Array:

```
java.lang.reflect.Array
java.sql.Array
```

Oct 5, 2022

Sprenkle - CSCI209

19

19

Packaging Code

- Use package keyword to say that a class belongs to a package:
 - **package** my.package.name;
 - First line in class file
 - Classes without a declared package (like what we've been doing) are in the default package
- Typically, use a unique prefix, similar to domain names
 - com.ibm
 - edu.wlu.cs.logic
- Use package name to create directory hierarchy
 - For example, code in edu.wlu.cs.logic package would be in a logic directory inside a cs directory inside a wlu directory inside a edu directory

We will start organizing our code in packages soon...

Oct 5, 2022

Sprenkle - CSCI209

20

20

INHERITANCE

Oct 5, 2022

Sprenkle - CSCI209

21

21

Review: Inheritance (from CSCI112)

- What are the benefits of inheritance?
- What are examples of inheritance?
- When should you use inheritance?

Oct 5, 2022

Sprenkle - CSCI209

22

22

Inheritance

- Build new classes based on existing classes
 - Allows *code reuse*
- Start with a class (**parent** or **super class**)
- Create another class that extends or *specializes* the class
 - Called **the child, subclass, or derived class**
 - Use **extends** keyword to make a subclass

Oct 5, 2022

Sprenkle - CSCI209

23

23

Child class

- Inherits all of parent class's methods and fields
 - Note on **private** fields: all are *inherited*, just can't *access*
- Constructors are **not** inherited
- Can **override** methods
 - Recall: *overriding* - methods have the same name and parameters, but implementation is different
- Can add methods or fields for *additional functionality*
- Use **super** object to call parent's method
 - Even if child class redefines parent class's method

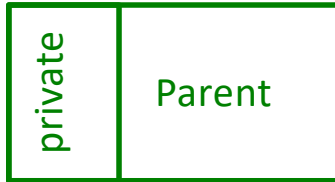
Oct 5, 2022

Sprenkle - CSCI209

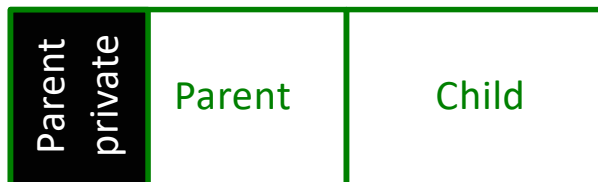
24

24

Inheriting Private Variables



Parent has private variables.
Objects of Parent class can access.



Child class inherits the private variables from Parent but cannot *directly* access them. Call Parent methods that can!

Oct 5, 2022

Sprenkle - CSCI209

25

25

Rooster class

- Could write class from scratch, but ...
- A rooster *is a* chicken
 - But it adds something to (or *specializes*) what a chicken is/does
- Classic mark of inheritance: *is a* relationship
- Rooster is child class
- Chicken is parent class

Oct 5, 2022

Sprenkle - CSCI209

26

26

Access Modifiers

● public

- Any class can access

● private

- No other class can access (including child classes)
 - Must use parent class's public accessor/mutator methods

● protected



- Child classes can access
- Members of package can access
- Other classes cannot access

Oct 5, 2022

Sprenkle - CSCI209

27

27

Access Modes

Default (if none specified)

Accessible to	Member Visibility			
	public	protected	package	private
Defining class	Yes	Yes	Yes	Yes
Class in same package	Yes	Yes	Yes	No
Subclass in different package	Yes	Yes	No	No
Non-subclass different package	Yes	No	No	No

- Visibility for fields: who can *access/change*
- Visibility for methods: who can *call*

Oct 5, 2022

Sprenkle - CSCI209

28

28

protected

- Accessible to subclasses and members of package
- Can't keep encapsulation "pure"
 - Don't want others to access fields directly
 - May break code if you change your implementation
- Assumption?
 - Someone extending your class with protected access (or in same package) knows what they are doing

Oct 5, 2022

Sprenkle - CSCI209

29

29

Guidance on Access Modifiers

- If you're uncertain which access modifier to use (public, protected, package/default, or private), use the *most restrictive*
 - Changing to less restrictive later → easy
 - Changing to more restrictive → may break code that uses your classes

Oct 5, 2022

Sprenkle - CSCI209

30

30

Looking Ahead: Exam on Friday

- Exam released at 8 a.m. on Friday
 - Closes: Sunday at 11:59 p.m.
 - No Class on Friday
 - I am available for office 1:30-2:30 (but not during 8:30-9:30)
- Online, timed exam: 70 minutes
- Open book/notes/slides – but **do not** rely on that
 - NOT open internet
- Prep document online
- 3 sections:
 - Very Short Answer, Short Answer, Coding