

Objectives

- Polymorphism
- Inheritance
 - Final methods, fields
- Eclipse

Sept 19, 2008

Sprenkle - CS209

1

Assignment 4

- Increase to worth 100 points?

Sept 19, 2008

Sprenkle - CS209

2

Review

- How do we refer to the parent class?
- What are the access modes, ordered from least restrictive to most restrictive?

Sept 19, 2008

Sprenkle - CS209

3

Polymorphism

- You can use a child class object whenever the program expects an object of the parent class
- Object variables are *polymorphic*
- A Chicken object variable can refer to an object of class Chicken, Hen, Rooster, or any class that inherits from Chicken

Sept 19, 2008

Sprenkle - CS209

4

Polymorphism

```
Chicken[] chickens = new Chicken[3];
chickens[0] = momma;
chickens[1] = foghorn;
chickens[2] = baby;
```

- We can guess the actual types but compiler can't

Sept 19, 2008

Sprenkle - CS209

5

Polymorphism

```
Chicken[] chickens = new Chicken[3];
chickens[0] = momma;
chickens[1] = foghorn;
chickens[2] = baby;
```

- We think `chicken[1]` is probably a Rooster, but to *compiler*, it's a Chicken so
`chicken[1].crow();`
 will not compile

Sept 19, 2008

Sprenkle - CS209

6

Polymorphism

- When we refer to a Rooster object through a Rooster object variable, we see it as a Rooster object
- If we refer to a Rooster object through a Chicken object variable, we see it as a Chicken object.
- We cannot assign a parent class object to a derived class object variable
 - A Rooster is a Chicken, but a Chicken is not necessarily a Rooster

Sept 19, 2008

Sprenkle - CS209

7

Polymorphism

- Which method do we call if we call `chicken[1].feed()`
Rooster's or Chicken's?

Sept 19, 2008

Sprenkle - CS209

8

Polymorphism

- Which method do we call if we call `chicken[1].feed()`
Rooster's or Chicken's?
- Rooster's!
 - Object is a Rooster
 - The JVM figures out its class at runtime and runs the appropriate method
- **Dynamic dispatch**
 - At runtime, the class of the object is determined. Then, the appropriate method for that class is dispatched

Sept 19, 2008

Sprenkle - CS209

9

Dynamic vs. Static Dispatch

- Dynamic dispatch is a property of Java, not object-oriented programming in general
- Some OOP languages use **static dispatch** where the type of the object variable used to call the method determines which version gets run
- The primary difference is **when decision on which method to call is made...**
 - Static dispatch (C#) decides at compile time
 - Dynamic dispatch (Java, Python) decides at run time

Sept 19, 2008

Sprenkle - CS209

10

Feed the Chickens!

```
for( Chicken c: chickens ) {
    c.feed();           How to read this code?
}
```

- Dynamic dispatch calls the appropriate method in each case, corresponding to the actual class of each object.
 - This is the power of polymorphism and dynamic dispatch!

Sept 19, 2008

Sprenkle - CS209

11

Preventing Inheritance

- Sometimes, you do not want a class to derive from one of your classes.
- A class that cannot be extended is known as a **final** class.
- To make a class final, simply add the keyword **final** in front of the class definition:

```
final class Rooster extends Chicken {
    . . .
}
```

Sept 19, 2008

Sprenkle - CS209

12

Final methods

- Can make a method **final**
 - Any class derived from this class cannot override the **final** methods
- By default, all methods in a **final** class are **final** methods.

```
class Chicken {
    . . .
    public final String getname() { . . . }
    . . .
}
```

Sept 19, 2008

Sprenkle - CS209

13

Why **final** methods and classes?

- **Efficiency**
 - Compiler can replace a **final** method call with an inline method
 - Does not have to worry about another form of this method that belongs to a derived class
 - JVM does not need to determine which method to call dynamically
- **Safety**
 - No alternate form of the method; straightforward which version of the method you called.
- Example of **final** class: **System**

Sept 19, 2008

Sprenkle - CS209

14

Explicit Object Casting

- Just like we can cast variables:

```
double pi = 3.14; int i_pi = (int) pi;
```
- We can cast objects.

```
Rooster foghorn = (Rooster)chickens[1];
```

 - Use casting to use an object in its full capacity after its actual type (the derived class) has been forgotten

Sept 19, 2008

Sprenkle - CS209

15

Example: Explicit Object Casting

- Rooster object is referred to only using a Chicken object variable
 - `chickens[1]` refers to an object variable to a Chicken object
 - We cannot access any of the Rooster-specific fields or methods using this object variable.
- Create new object variable to Rooster object
 - Using this variable allows us to reference the Rooster-specific fields...

```
Rooster rooster = (Rooster) chickens[1];
```

Sept 19, 2008

Sprenkle - CS209

16

Object Casting

- We can do explicit type casting because `chickens[1]` refers to an object that is actually a Rooster object.
- For example, cannot do this with `chickens[0]` because it refers to a Hen (not Rooster) object

```
Rooster rooster = (Rooster) chickens[1];
// OK; chickens[1] refers to a Rooster object
Rooster hen = (Rooster) chickens[0];
// ERROR; chickens[1] refers to a Hen object
```

- Promising the compiler that `chickens[1]` really refers to a Rooster object, although it is an object variable to a Chicken object
- If this is not the case, generates an exception
 - More about exceptions later

Sept 19, 2008

Sprenkle - CS209

17

instanceof Operator

- Use the **instanceof** operator to make sure such a cast will succeed

```
if (chickens[1] instanceof Rooster) {
    rooster = (Rooster)chickens[1];
}
```

- Operator returns a **boolean**
 - true iff `chickens[1]` refers to an object of type Rooster
 - false otherwise

Update Chicken class

Sept 19, 2008

Sprenkle - CS209

18

Summary of Inheritance

- Place common operations & fields in parent class
 - Remove repetitive code by modeling the “is-a” hierarchy
 - Move “common denominator” code up the inheritance chain
- Don’t use inheritance unless *all* inherited methods make sense
- Use polymorphism

Sept 19, 2008

Sprenkle - CS209

19

JAVADOCS

Sept 19, 2008

Sprenkle - CS209

20

Javadocs

- Special comments, which are used to generate HTML documentation
- Syntax:


```
/**
 * Comment
 */
```
- Put before a class, a method, or a field to describe the respective class/method/field

Sept 19, 2008

Sprenkle - CS209

21

Javadoc

- Can contain HTML syntax in description
- Example block comments to help describe your code


```
@param <paramname> <description>
@return <description> (include special cases)
@author <author's name>
```
- More that we'll get to later

Sept 19, 2008

Sprenkle - CS209

22

Examples

```
/**
 * A simple Java class that models a Chicken. The
 * state of the chicken is its name, height, and weight
 *
 * @author Sara Sprenkle
 */

/**
 * @return the height of the chicken, in centimeters
 */

/**
 * @param n the String representing the name of the
 * chicken
 */
```

Sept 19, 2008

Sprenkle - CS209

23

ECLIPSE

Sept 19, 2008

Sprenkle - CS209

24

Eclipse

- Open source integrated development environment (IDE) for Java
- Has market share for Java IDEs
- Described as “an open extensible IDE for anything and nothing in particular”
- Provides a robust Java development environment
- Incorporates popular software development tools like JUnit and CVS
 - More on those later this semester
- Plugins allow extensibility

Sept 19, 2008

Sprenkle - CS209

25

Project/Code Organization

- **workspace** directory contains all projects
 - Located in your home directory, unless you specified otherwise
- Use **projects** to organize your code
- Within a project
 - `src/` directory contains `.java` files
 - `bin/` directory contains `.class` files

Sept 19, 2008

Sprenkle - CS209

26

Java Made Easier

- Creating class's basic functionality
 - See **Source** and **Refactor** menus
- Gives you a list of methods for an object
 - After you type `object.`
 - Then shows parameters for methods
- Automatically creates template of Javadoc
 - When you type `/**`

Sept 19, 2008

Sprenkle - CS209

27

Assignment 5

- Using Eclipse
- Creating an online library
 - 5 classes of objects
 - Driver program
- Due on Wednesday
 - Another small (50 point) assignment on Monday, due on Wednesday

Sept 19, 2008

Sprenkle - CS209

28