

## Objectives

- Parameter passing in Java
- Inheritance

Sept 17, 2008

Sprenkle - CS209

1

## Review

- What does **static** mean?
- When should a method be declared **static**?

Sept 17, 2008

Sprenkle - CS209

2

## Method Parameters in Java

- Java always passes parameters into methods **by value**
  - Methods cannot change the variables used as input parameters
  - This is a subtle point so we need to go through several examples
- Python is something that's not quite pass-by-value--it depends on if the object is mutable or immutable
  - *Pass-by-alias* is one term used

Sept 17, 2008

Sprenkle - CS209

3

## Method Parameters

```
public static void main(String[] args) {
    int x = 10;
    int squared = square(x);
    System.out.println("The square of " + x + " is " + squared);
}

public static int square(int num) {
    return num*num;
}
```

Draw the stack as it changes (similar to Python)

main	x	10
------	---	----

Sept 17, 2008

Sprenkle - CS209

4

## What's the output?

```
int x = 27;
System.out.println(x);
doubleValue(x);
System.out.println(x);
```

...

```
void doubleValue(int p) {
    p = p * 2;
}
```

Sept 17, 2008

Sprenkle - CS209

5

## What's the output?

```
int x = 27;
System.out.println(x);
doubleValue(x);
System.out.println(x);
```

...

```
void doubleValue(int p) {
    p = p * 2;
}
```

27  
27

Sept 17, 2008

Sprenkle - CS209

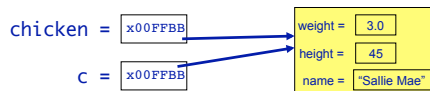
6

## Pass by Value: Objects

- Primitive types are a little more obvious
  - Can't change original variable
- For objects, passing a copy of the parameter looks like

```
public void methodName(Chicken c)
```

```
methodName(chicken);
```



Sept 17, 2008

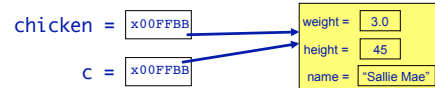
Sprenkle - CS209

7

## Pass by Value: Objects

- What does this mean?

```
methodName(chicken);
```



```
public void methodName(Chicken c) {
    if( c.getWeight() < MIN ) {
        c.feed();
    }
    ...
}
```

Does chicken  
change in calling  
method?

Sept 17, 2008

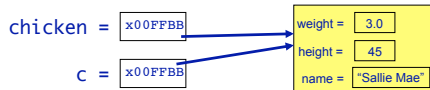
Sprenkle - CS209

8

## Pass by Value: Objects

- What does this mean?

```
methodName(chicken);
```



```
public void methodName(Chicken c) {
    if( c.getWeight() < MIN ) {
        c.feed();
    }
    ...
}
```

Does chicken  
change in calling  
method? YES!  
Both chicken and c  
are pointing to the  
same object

Sept 17, 2008

Sprenkle - CS209

9

## What's the output?

```
Farm farm = new Farm("OldMac");
Chicken sal = new Chicken("Sallie Mae", 50, 10);
System.out.println(sal.getWeight());
farm.feedChicken(sal);
System.out.println(sal.getWeight());
...
```

```
void feedChicken(Chicken c) {
    c.setWeight( c.getWeight() + .5);
}
```

Sept 17, 2008

Sprenkle - CS209

10

## What's the output?

```
Farm farm = new Farm("OldMac");
Chicken sal = new Chicken("Sallie Mae", 50, 10);
System.out.println(sal.getWeight());
farm.feedChicken(sal);
System.out.println(sal.getWeight());
...
```

```
void feedChicken(Chicken c) {
    c = new Chicken(c.getName(), c.getWeight(),
        c.getHeight());
    c.setWeight( c.getWeight() + .5);
}
```

Sept 17, 2008

Sprenkle - CS209

11

## What's the Difference?

```
Farm farm = new Farm("OldMac");
Chicken sal = new Chicken("Sallie Mae", 50, 10);
System.out.println(sal.getWeight());
farm.feedChicken(sal);
System.out.println(sal.getWeight());
...
```

```
void feedChicken(Chicken c) {
    c = new Chicken(c.getName(), c.getWeight(),
        c.getHeight());
    c.setWeight( c.getWeight() + .5);
}
```



Sept 17, 2008

Sprenkle - CS209

12

## What's the Difference?

```
void feedChicken(Chicken c) {
    c = new Chicken(c.getName(), c.getWeight(),
        c.getHeight());
    c.setWeight( c.getWeight() + .5);
}
```



Sept 17, 2008

Sprenkle - CS209

13

## Summary of Method Parameters

- Everything is passed **by value** in Java
- 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

Sept 17, 2008

Sprenkle - CS209

14

## INHERITANCE

Sept 17, 2008

Sprenkle - CS209

15

## Inheritance

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

Sept 17, 2008

Sprenkle - CS209

16

## Child class

- Inherits all of parent class's methods and fields
  - Unless they're **private** or **static**
- Can also **override** methods
  - Use the same name, but the implementation is different
- Adds methods or fields for additional functionality
- If the child class redefines a parent class's method, can still call the parent class's method on the **super** object

Sept 17, 2008

Sprenkle - CS209

17

## Inheritance

- Access modifiers in child classes
  - Can make access to child class **more** restrictive but not less restrictive
- Class fields and methods are **not** inherited
- Constructors are **not** inherited
  - We had to define `Rooster( String name, int height, double weight)` even though similar constructor in `Chicken`

Sept 17, 2008

Sprenkle - CS209

18

## 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
- The *is a* relationship
  - Classic mark of inheritance
- Rooster is child class
- Chicken is parent class

Sept 17, 2008

Sprenkle - CS209

19

## Modify Chicken Class

- Want instance variables to be accessible by child class
  - Can't be *private*
- Add new boolean instance variable `is_female`

Sept 17, 2008

Sprenkle - CS209

20

## Access Modifiers

- *public*
  - Any class can access
- *private*
  - No other class can access (including child classes)
    - Must use parent class's accessor/mutator methods
- *protected*
  - Child classes can access
  - Members of package can access
  - Other classes cannot access

Sept 17, 2008

Sprenkle - CS209

21

## 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

Which access modifier should we use for the *Chicken* instance variables?

Sept 17, 2008

Sprenkle - CS209

22

## 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 knows what they are doing

Sept 17, 2008

Sprenkle - CS209

23

## Access Modifiers

- If you're uncertain which to use (protected, package, or private), use the most restrictive
  - Changing to less restrictive later is easy
  - Changing to more restrictive may break the code that uses your classes.

Sept 17, 2008

Sprenkle - CS209

24

## Look at Modified Chicken Class

Sept 17, 2008

Sprenkle - CS209

25

## Rooster class

```
public class Rooster extends Chicken {
    public Rooster( String name,
        int height, double weight) {
        // all instance fields inherited
        // from super class
        this.name = name;
        this.height = height;
        this.weight = weight;
        is_female = false;
    }

    // new functionality
    public void crow() { ... }
    ...
}
```

By default calls default  
super constructor  
with no parameters

Sept 17, 2008

Sprenkle - CS209

26

## Rooster class

```
public class Rooster extends Chicken {
    public Rooster( String name,
        int height, double weight) {

        super(name, height, weight);
        is_female = false;
    }

    // new functionality
    public void crow() { ... }

    ...
}
```

Call to super constructor must be first line in constructor

Sept 17, 2008

Sprenkle - CS209

27

## Constructor Chaining

- Constructor automatically calls constructor of parent class if not done explicitly
  - `super();`
- What if parent class does not have a constructor with no parameters?
  - Compilation error
  - Forces subclasses to call a constructor with parameters

Sept 17, 2008

Sprenkle - CS209

28

## Overriding Methods in Parent Class

```
public class Rooster extends Chicken {
    ...

    // new functionality
    public void crow() {
        System.out.println("Cock-a-Doodle-Do!");
    }

    // overrides superclass; greater gains
    public void feed() {
        weight += .5;
        height += 2;
    }
}
```

Sept 17, 2008

Sprenkle - CS209

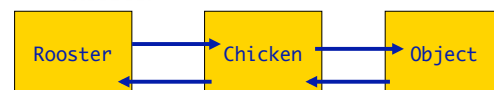
29

## Inheritance Tree

- `java.lang.Object`

- `Chicken`

- `Rooster`



- Call constructor of parent class first
  - Know you have the fields of parent class before you implement constructor for your class

Sept 17, 2008

Sprenkle - CS209

30

## Inheritance Tree

- `java.lang.Object`

- `Chicken`

- `Rooster`



- No `finalize()` chaining

- Should call `super.finalize()` inside of `finalize` method

Sept 17, 2008

Sprenkle - CS209

31

## Shadowing Parent Class Fields

- Child class has field with same name as parent class

- You probably shouldn't be doing this!

- But could happen

- Possibly: more precision for a constant

```

field          // this class's field
this.field     // this class's field
super.field    // super class's field
  
```

Sept 17, 2008

Sprenkle - CS209

32

## Multiple Inheritance

- In Python, it is possible for a class to inherit (or extend) more than one parent class

- The child class has the fields from both parent classes

- This is NOT possible in Java.

- A class may extend (or inherit from) only one class

Sept 17, 2008

Sprenkle - CS209

33

## Assignment 4

- Start of a simple video game

- Game class to run

- `GameObject` is parent class of other moving objects

- Some poor design

- Can't fix until see other Java structures

- Don't need to understand all of the code, just some of it

- Create a `Goblin` class and a `Treasure` class

- Move `Goblin` and `Treasure`

Sept 17, 2008

Sprenkle - CS209

34