Lesson 15: OOP #5: Overriding Methods (W04D3)

Balboa High School

Michael Ferraro

September 11, 2015

Do Now

Answer the following (in a text editor):

1. T/F: A subclass inherits the instance variables (fields) of its superclass. E.g., `Teacher` has the fields `age` and `greeting` from its superclass, `Person`.

2. If we add a field variable to `Teacher` called `roomNumber` and add a method that sets to value of `roomNumber` — without returning any information — then the method is a(n) _____ method.

3. Consider a new kind of person, a *baker*. If class `Baker` has superclass `Person`, which of the following is the correct class declaration?

   a. `public class Person extendsInto Baker { ...}`
   b. `public class Person extendsFrom Baker { ...}`
   c. `public class Baker extends Person { ...}`
   d. `public class Person extends Baker { ...}`

Aim

Students will finish learning about *class inheritance* and make progress on Problem Set #2.

Recap of Last Class

- We created class `Teacher`, a subclass of `Person`

Recap of Last Class

- We created class `Teacher`, a subclass of `Person`
- Subclasses inherit instance vars/fields and methods of superclass; an example of *code reuse*

Recap of Last Class

- We created class `Teacher`, a subclass of `Person`
- Subclasses inherit instance vars/fields and methods of superclass; an example of *code reuse*
- We added methods to `Teacher`, making it a *richer* class than `Person`

Recap of Last Class

- We created class `Teacher`, a subclass of `Person`
- Subclasses inherit instance vars/fields and methods of superclass; an example of *code reuse*
- We added methods to `Teacher`, making it a *richer* class than `Person`
- We continued to follow good class design practice: The instance vars of `Teacher` are *private*, meaning that access to the field values are managed by *accessor* and *mutator* methods

Angry Persons

1. Add a method to the `Person` class called `makeAngry()`. The method
   - takes no arguments (input values)
   - returns no data
   - prints this to the console:

     `You have made me, <`*`fullName`*`>, angry.  I will get even.`
2. Once your method is written, update your driver so that `ralph` and `teacher1` get angry.

Overriding a Superclass' Methods

- When you make a `Teacher` angry, Java looks in the `Teacher` class for the `makeAngry()` method; it finds none and next looks to the superclass (`Person`) and finds the method.
- But let's say that a `Teacher` doesn't get *angry* in the same way a normal person does.
- In `Teacher.java`, add a method with the same name as the one you added to `Person.java` — `makeAngry()`. Make this method print this message to the console:

  `You have made your teacher, <`*`fullName`*`>, angry. Your grade has been reduced by 6%.`

Overriding a Superclass' Methods

Problem: Even though `Teacher` is a subclass of `Person`, it cannot see the variable `fullName` since it's `private`! When you write `makeAngry()` in the `Teacher class`, you must get the `Teacher`'s fullName just like everyone else:

`<object_name>.getFullName()`

But when you're writing code inside `Teacher.java`, you don't have knowledge of the instance's name — you can't exactly say "`teacher1.getFullName()`".

Overriding a Superclass' Methods

Problem: Even though `Teacher` is a subclass of `Person`, it cannot see the variable `fullName` since it's `private`! When you write `makeAngry()` in the `Teacher class`, you must get the `Teacher`'s fullName just like everyone else:

<*object_name*>`.getFullName()`

But when you're writing code inside `Teacher.java`, you don't have knowledge of the instance's name — you can't exactly say "`teacher1.getFullName()`".

Solution: The keyword `this` always refers to the current instance of an object.

Correct makeAngry() Method

In class Teacher:

```
public void makeAngry() {
    System.out.println("You have made your teacher, "
      + this.getFullName()
      + ", angry.  Your grade has been reduced by 6%.");
}
```

Another Correct `makeAngry()` Method

In class `Teacher`:

```
public void makeAngry() {
    System.out.println("You have made your teacher, "
      + getFullName()
      + ", angry.  Your grade has been reduced by 6%.");
}
```

For the rest of the period...

- Write the `Student` class
  - Include the `Student`-specific field variables we discussed earlier
    - make fields `private`
    - write accessor & mutator methods for all fields
  - Override the `makeAngry()` method (you choose what happens when you make a student *angry*).
  - Instantiate `Student` as `student1`. Show its field values and make it angry.
- Quiz #2 coming up in 2 class days!
- Continue working on PS #2.

HW

Continue working on PS #2. You should be able to complete some or all of §6 by now. **If you're not done with §§1-5 yet, start coming in for help!**