CSE 143: Computer Programming II

Inheritance & Polymorphism

Today's Goals

1. Time!
   - Clock
     - DigitalClock
     - PreciseDigitalClock
     - AnalogClock

   Our goals are to understand how methods get inherited and how objects in a hierarchy interact:
   - Clock c = new DigitalClock(true);
   - AnalogClock ac = new DigitalClock(true);
   - PreciseDigitalClock pdc = new DigitalClock(true);
   - c.getTime(); ac.getTime(); pdc.getTime();

Clock Class

1. public class Clock {
   2.     private int hour;
   3.     private int minute;
   4. }

   OUTPUT
   >>> Clock c = new Clock(); // hour = 4, minute = 12
   >>> System.out.println(c.getTime() + "..." + c.getHour() + "..." + c.getMinute());
   >>> 4 12...4...12

   What specializations could we make to Clock?
   - An “analog” clock with a face?
   - A “digital” clock with military time?
   - A clock with seconds?

Clock Hierarchy

For each of the following, is it always, sometimes, or never true:
- A DigitalClock is a Clock?
- An AnalogClock is a DigitalClock?
- A PreciseDigitalClock is a DigitalClock?
- A DigitalClock is a PreciseDigitalClock?
- A Clock is a DigitalClock?
- A Clock is an AnalogClock?

AnalogClock Class

1. public class AnalogClock extends Clock {
   2.     public static final int NUM_HOURS = 12;
   3.     public static final int NUM_MINUTES = 60;
   4. }

   OUTPUT
   >>> Clock c = new Clock(); // hour = 4, minute = 12
   >>> System.out.println(c.getTime() + "..." + c.getHour() + "..." + c.getMinute());
   >>> 4 12...4...12

   AnalogClock vs. Clock
   - Is an AnalogClock a Clock?
     Always! An AnalogClock is a Clock with extra features.

   What is different about an AnalogClock?
   - It has new methods: getHourHandAngle, getMinuteHandAngle
   - It “overrides” getTime to do something different
This doesn't compile. Clock doesn't have a getPeriod method!

Notice that Java knows that DigitalClock is a Clock with extra features. If we remove the second and third lines, we get:

This doesn't compile. Clock doesn't have a getSecond method!
Now, we do the same idea with a mystery problem!
Mystery Problem #4

Snow
created: method2()
created: method3()

Rain
created: method1()
overriden: method2()
inherited: method3()

Sleet
overriden: method2()
overriden: method3()

Fog
created: method1()
inherited: method2()
overriden: method3()

var2 is a var2 restricted to a

var2
new Rain();
var2.method2();

OUTPUT
>> Rain 2

Mystery Problem #5

Snow
created: method2()
created: method3()

Rain
created: method1()
overriden: method2()
inherited: method3()

Sleet
overriden: method2()
overriden: method3()

Fog
created: method1()
inherited: method2()
overriden: method3()

var2 is a var2 restricted to a

var2
new Rain();
((Sleet) var2).method2();

OUTPUT
>> ClassCastException: *Error*

Mystery Problem #6

Snow
created: method2()
created: method3()

Rain
created: method1()
overriden: method2()
inherited: method3()

Sleet
overriden: method2()
overriden: method3()

Fog
created: method1()
inherited: method2()
overriden: method3()

var2 is a var2 restricted to a

var2
new Fog();
((Sleet)var2).method2();

OUTPUT
>> Sleet 2
>> Snow 2
>> Fog 3