Today's Goals

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 Hierarchy

For each of the following, is it always, sometimes, or never true:
- A DigitalClock is a Clock?
- Always! A DigitalClock is a type of Clock with digital features.
- An AnalogClock is a DigitalClock?
- Sometimes! Not all DigitalClocks have seconds, but those that do are PreciseDigitalClocks.
- A PreciseDigitalClock is a DigitalClock?
- Always! A PreciseDigitalClock is a DigitalClock that includes seconds.
- A DigitalClock is a PreciseDigitalClock?
- Sometimes! Not all DigitalClocks have seconds, but those that do are PreciseDigitalClocks.
- A Clock is a DigitalClock?
- Sometimes! Not all Clocks have DigitalClock features, but those that do are DigitalClocks.

AnalogClock Class

For each of the following, is it always, sometimes, or never true:
- 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
Public class AnalogClock extends Clock {
    public static final int NUM_HOURS = 12;
    ... System.out.println(c1.getMinuteHandAngle());
OUTPUT
>> Hour Hand: 180%, MinuteHand: 60%
>> 180
>> 60
>> 55
>> 30
>> 0

This last one is a compilation error. (DigitalClock doesn't have a getSecond() method)
DigitalClock Puzzles (Continued, Continued) 11

DigitalClock Puzzle #5

public class DigitalClock {
    //hour=13, minute=22
    System.out.println(PreciseDigitalClock.c6.getTime());
    System.out.println(PreciseDigitalClock.c6.getSecond());
}

All of these are ClassCastExceptions. A new DigitalClock() is NOT a PreciseDigitalClock

Now, we do the same idea with a mystery problem!

Mystery Problem #1

1 public class Snow {
2     public void method2() {
3         System.out.println("Snow 2");
4     }
5     public void method3() {
6         System.out.println("Snow 3");
7     }
8 }

Class Diagram

var2 is a Snow
Sleet is a Snow
Rain is a Snow

var2
created: method2()
created: method3()

var2
overriden: method2()
overriden: method3()

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

Mystery Problem #2

5 public class Snow {
6     public void method2() {
7         System.out.println("Snow 2");
8     }
9     public void method3() {
10        System.out.println("Snow 3");
11     }
12 }

Class Diagram

var2
created: method2()

Snow
Fog
Sleet
Rain

var2
overriden: method2()
overriden: method3()

Snow var2 = new Rain();
var2.method1();
OUTPUT
>> Rain 2
>> Rain 2
>> Snow 2
>> Sleet 2
>> Snow 2
>> Sleet 3
>> Rain 2

Mystery Problem #3

5 public class Snow {
6     public void method2() {
7         System.out.println("Snow 2");
8     }
9     public void method3() {
10        System.out.println("Snow 3");
11     }
12 }

Class Diagram

var2
created: method2()

Snow
Fog
Sleet
Rain

var2
overriden: method2()
overriden: method3()

Snow var2 = new Rain();
var2.method1();
OUTPUT
>> Rain 1
>> Rain 1

Mystery Problems

Keep the following rules in mind
- If the type on the left doesn’t have a method, we can’t call it.
- When calling a method, the version called is always the actual type.
- Casting up the tree is the only type that is okay.

What do each of the following do? (error? print what?)

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

Snow var2 = new Fog();
((Sleet)var2).method2();
Snow var2 = new Rain();
((Rain)var2).method1();

Polymorphism Mystery 12

public class Rain extends Snow {
    public void method1() {
        System.out.println("Rain 1");
    }
    public void method2() {
        System.out.println("Rain 2");
    }
    public void method3() {
        System.out.println("Rain 3");
    }
}

public class Sleet extends Snow {
    public void method1() {
        System.out.println("Sleet 1");
    }
    public void method2() {
        System.out.println("Sleet 2");
    }
    public void method3() {
        System.out.println("Sleet 3");
    }
}

PUBLIC Class Diagram

Rain is a Snow

var2
created: method1()
overriden: method2()
overriden: method3()

Sleet
created: method1()
overriden: method2()
overriden: method3()

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

Output
>> Sleet 2
>> Snow 2
>> Sleet 3

Mystery Problem #3

5 public class Snow {
6     public void method2() {
7         System.out.println("Snow 2");
8     }
9     public void method3() {
10        System.out.println("Snow 3");
11     }
12 }

Class Diagram

var2
created: method2()

Snow
Fog
Sleet
Rain

var2
overriden: method2()
overriden: method3()

Snow var2 = new Rain();
var2.method1();
OUTPUT
>> Rain 1
>> Rain 1

Mystery Problem #3

5 public class Snow {
6     public void method2() {
7         System.out.println("Snow 2");
8     }
9     public void method3() {
10        System.out.println("Snow 3");
11     }
12 }

Class Diagram

var2
created: method2()

Snow
Fog
Sleet
Rain

var2
overriden: method2()
overriden: method3()

Snow var2 = new Rain();
var2.method1();
OUTPUT
>> Rain 1
>> Rain 1

Mystery Problem #2

5 public class Snow {
6     public void method2() {
7         System.out.println("Snow 2");
8     }
9     public void method3() {
10        System.out.println("Snow 3");
11     }
12 }

Class Diagram

var2
created: method2()

Snow
Fog
Sleet
Rain

var2
overriden: method2()
overriden: method3()

Snow var2 = new Rain();
var2.method1();
OUTPUT
>> Rain 2
>> Rain 2
>> Snow 2
>> Sleet 2
>> Snow 2
>> Sleet 3
>> Rain 2

Mystery Problem #3

5 public class Snow {
6     public void method2() {
7         System.out.println("Snow 2");
8     }
9     public void method3() {
10        System.out.println("Snow 3");
11     }
12 }

Class Diagram

var2
created: method2()

Snow
Fog
Sleet
Rain

var2
overriden: method2()
overriden: method3()

Snow var2 = new Rain();
var2.method1();
OUTPUT
>> Rain 1
>> Rain 1

Mystery Problem #2

5 public class Snow {
6     public void method2() {
7         System.out.println("Snow 2");
8     }
9     public void method3() {
10        System.out.println("Snow 3");
11     }
12 }

Class Diagram

var2
created: method2()

Snow
Fog
Sleet
Rain

var2
overriden: method2()
overriden: method3()

Snow var2 = new Rain();
var2.method1();
OUTPUT
>> Rain 2
>> Rain 2
>> Snow 2
>> Sleet 2
>> Snow 2
>> Sleet 3
>> Rain 2

Mystery Problem #3

5 public class Snow {
6     public void method2() {
7         System.out.println("Snow 2");
8     }
9     public void method3() {
10        System.out.println("Snow 3");
11     }
12 }

Class Diagram

var2
created: method2()

Snow
Fog
Sleet
Rain

var2
overriden: method2()
overriden: method3()

Snow var2 = new Rain();
var2.method1();
OUTPUT
>> Rain 1
>> Rain 1

Mystery Problem #2

5 public class Snow {
6     public void method2() {
7         System.out.println("Snow 2");
8     }
9     public void method3() {
10        System.out.println("Snow 3");
11     }
12 }

Class Diagram

var2
created: method2()

Snow
Fog
Sleet
Rain

var2
overriden: method2()
overriden: method3()

Snow var2 = new Rain();
var2.method1();
OUTPUT
>> Rain 2
>> Rain 2
>> Snow 2
>> Sleet 2
>> Snow 2
>> Sleet 3
>> Rain 2

Mystery Problem #3

5 public class Snow {
6     public void method2() {
7         System.out.println("Snow 2");
8     }
9     public void method3() {
10        System.out.println("Snow 3");
11     }
12 }

Class Diagram

var2
created: method2()

Snow
Fog
Sleet
Rain

var2
overriden: method2()
overriden: method3()

Snow var2 = new Rain();
var2.method1();
OUTPUT
>> Rain 1
>> Rain 1
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
restricted
to a

Snow
Rain
Sleet
Fog

var2
is 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
restricted
to a

Snow
Rain
Sleet
Fog

var2
is 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
restricted
to a

Snow
Rain
Sleet
Fog

var2
is a

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

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