

CSE 143

Computer Programming II

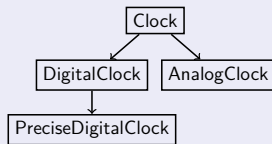
Inheritance & Polymorphism



Today's Goals

1

Time!



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

2

Clock Class

```

1 public class Clock {
2     private int hour;
3     private int minute;
4
5     public int getMinute() { return this.minute; }
6     public int getHour() { return this.hour; }
7     public String getTime() { return hour + " " + minute; }
8 }
  
```

```

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

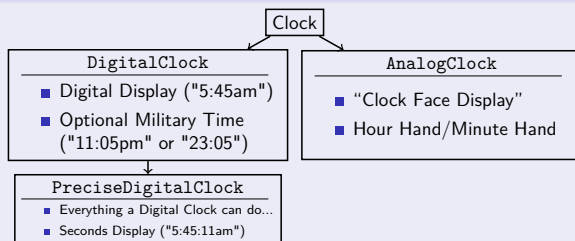
What specializations could we make to Clock?

- An "analog" clock with a face?
- A "digital" clock with military time?
- A clock with seconds?

Clock Hierarchy

3

Clock Hierarchy Diagram



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`?
Never! `AnalogClock`'s have a face; `DigitalClock`'s don't.
- A `PreciseDigitalClock` is a `DigitalClock`?
Always! A `PreciseDigitalClock` is a `DigitalClock` that includes seconds.
- A `DigitalClock` is a `PreciseDigitalClock`?
Sometimes! Not all `DigitalClock`s have seconds, but those that do are `PreciseDigitalClock`s.
- A `Clock` is a `DigitalClock`?
Sometimes! Not all `Clock`s have `DigitalClock` features, but those that do are `DigitalClock`s.

AnalogClock Class

4

Class

```

1 public class AnalogClock extends Clock {
2     public static final int NUM_HOURS = 12;
3     public static final int NUM_MINUTES = 60;
4
5     public double getHourHandAngle() {
6         return 360 * ((double) (this.getHour() % 12) / NUM_HOURS);
7     }
8
9     public double getMinuteHandAngle() {
10        return 360 * ((double) this.getMinute() / NUM_MINUTES);
11    }
12
13    public String getTime() {
14        return "Hour Hand: " + this.getHourHandAngle() + "%, "
15            + "Minute Hand: " + this.getMinuteHandAngle() + "%";
16    }
17 }
  
```

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

Class

```

1 public class AnalogClock extends Clock {
2     public static final int NUM_HOURS = 12;
3     public static final int NUM_MINUTES = 60;
4
5     public double getHourHandAngle() {
6         return 360 * ((double) (this.getHour() % 12) / NUM_HOURS);
7     }
8
9     public double getMinuteHandAngle() {
10        return 360 * ((double) this.getMinute() / NUM_MINUTES);
11    }
12
13    public String getTime() {
14        return "Hour Hand: " + this.getHourHandAngle() + "%, "
15            + "Minute Hand: " + this.getMinuteHandAngle() + "%";
16    }
17 }

```

AnalogClock Puzzle #1

```

1 AnalogClock c1 = new AnalogClock();
2 System.out.println(c1.getTime());
3 System.out.println(c1.getHourHandAngle());
4 System.out.println(c1.getMinuteHandAngle());

```

```

OUTPUT
>> Hour Hand: 180%, MinuteHand: 60%
>> 180
>> 60

```

Class

```

1 public class AnalogClock extends Clock {
2     public static final int NUM_HOURS = 12;
3     public static final int NUM_MINUTES = 60;
4
5     public double getHourHandAngle() {
6         return 360 * ((double) (this.getHour() % 12) / NUM_HOURS);
7     }
8
9     public double getMinuteHandAngle() {
10        return 360 * ((double) this.getMinute() / NUM_MINUTES);
11    }
12
13    public String getTime() {
14        return "Hour Hand: " + this.getHourHandAngle() + "%, "
15            + "Minute Hand: " + this.getMinuteHandAngle() + "%";
16    }
17 }

```

AnalogClock Puzzle #2

```

1 Clock c2 = new AnalogClock();
2 System.out.println(c2.getTime());
3 System.out.println(c2.getHourHandAngle());
4 System.out.println(c2.getMinuteHandAngle());

```

This doesn't compile! Java treats c2 like a Clock. The second and third calls don't make sense for a clock. If we remove the second and third lines, we get:

```

OUTPUT
>> Hour Hand: 180%, MinuteHand: 60%

```

```

public class DigitalClock extends Clock {
    private boolean usingMilitaryTime;

    public DigitalClock(boolean usingMilitaryTime) {
        this.usingMilitaryTime = usingMilitaryTime;
    }
    public boolean isMilitaryTime() { return usingMilitaryTime; }
    public int getHour() {
        if (this.isMilitaryTime()) || super.getHour() <= 12) {
            return super.getHour();
        }
        else { return super.getHour() - 12; }
    }
    public String getPeriod() {
        if (this.isMilitaryTime()) { return ""; }
        else if (super.getHour() <= 12) { return "am"; }
        else { return "pm"; }
    }
    public String getTime() {
        return this.getHour() + ":" + this.getMinute() + this.getPeriod();
    }
}

public class PreciseDigitalClock extends DigitalClock {
    private int second;

    public PreciseDigitalClock() { super(false); }
    public int getSecond() { return this.second; }
    public String getTime() {
        return this.getHour() + ":" + this.getMinute() + ":" + this.getSecond()
            + this.getPeriod();
    }
}

```

Clock vs. DigitalClock vs. PreciseDigitalClock

■ Is a DigitalClock a Clock?

Always! A DigitalClock is a Clock with extra features.

■ Is a PreciseDigitalClock a DigitalClock?

Always! A PreciseDigitalClock is a DigitalClock with extra features.

■ What is different about a DigitalClock (from a Clock)?

- It has a new constructor
- It has a new field: usingMilitaryTime
- It has new methods: getPeriod, isMilitaryTime
- It "overrides" getTime and getHour to do something different

■ What is different about a PreciseDigitalClock (from a DigitalClock)?

- It is missing the one argument constructor
- It has a new field: second
- It has a new method: getSecond
- It "overrides" getTime to do something different

DigitalClock Puzzle #1

```

1 DigitalClock c3 = new DigitalClock(false); //hour = 13, minute = 22
2 System.out.println(c3.getTime());
3 System.out.println(c3.getHour());
4 System.out.println(c3.getMinute());
5 System.out.println(c3.getPeriod());

```

```

OUTPUT
>> 1:22pm
>> 1
>> 22
>> pm

```

DigitalClock Puzzle #2

```

1 Clock c4 = new DigitalClock(false); //hour = 13, minute = 22
2 System.out.println(c4.getTime());
3 System.out.println(c4.getHour());
4 System.out.println(c4.getMinute());
5 System.out.println(c4.getPeriod());

```

This doesn't compile. Clock doesn't have a getPeriod method!

DigitalClock Puzzle #3

```

1 Clock c4 = new DigitalClock(false); //hour = 13, minute = 22
2 System.out.println(c4.getTime());
3 System.out.println(c4.getHour());
4 System.out.println(c4.getMinute());

```

```

OUTPUT
>> 1:22pm
>> 1
>> 22

```

Notice that Java knows that c4 is actually a DigitalClock.

DigitalClock Puzzle #4

```

1 PreciseDigitalClock c5 = new PreciseDigitalClock(); //hour=13,minute=22,second=52
2 System.out.println(c5.getTime());
3 System.out.println(c5.getHour());
4 System.out.println(c5.getMinute());
5 System.out.println(c5.getSecond());
6 System.out.println((DigitalClock)c5.getTime());
7 System.out.println((DigitalClock)c5.getSecond());

```

```

OUTPUT
>> 1:22:52pm
>> 1
>> 22
>> 52
>> 1:22:52pm
>> This last one is a compilation error. (DigitalClock doesn't have a getSecond() method)

```

DigitalClock Puzzles (Continued, Continued)

11

DigitalClock Puzzle #5

```
1 DigitalClock c6 = new DigitalClock(); //hour=13,minute=22
2 System.out.println((PreciseDigitalClock)c6.getSecond());
3 System.out.println((PreciseDigitalClock)c6.getTime());
4 System.out.println((PreciseDigitalClock)c6.getSecond());
```

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

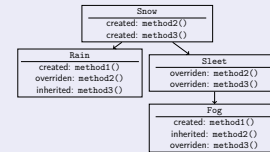
Now, we do the same idea with a **mystery** problem!

Polymorphism Mystery

12

```
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 }
9
10
11 public class Rain extends Snow {
12     public void method1() {
13         System.out.println("Rain 1");
14     }
15     public void method2() {
16         System.out.println("Rain 2");
17     }
18 }
19
20 public class Sleet extends Snow {
21     public void method2() {
22         System.out.println("Sleet 2");
23     }
24     super.method2();
25     this.method3();
26 }
27 public void method3() {
28     System.out.println("Sleet 3");
29 }
30 }
31
32 public class Fog extends Sleet {
33     public void method1() {
34         System.out.println("Fog 1");
35     }
36     public void method3() {
37         System.out.println("Fog 3");
38     }
39 }
40 }
```

Class Diagram



Mystery Problems

13

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

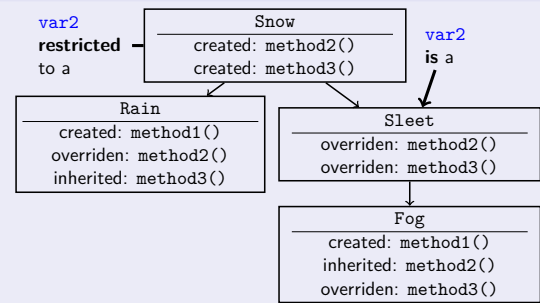
Snow var2 = new Fog();      Snow var2 = new Rain();
((Sleet)var2).method2();    var2.method2();

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

Mystery Problem #1

14

Class Diagram



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

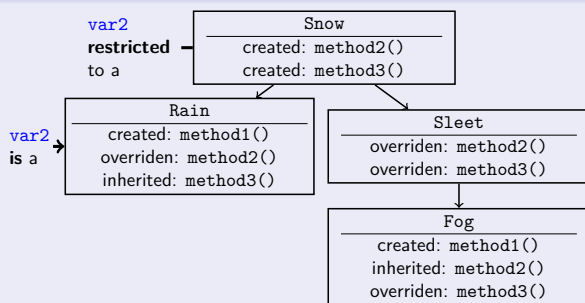
_____ OUTPUT _____

```
>> Sleet 2
>> Snow 2
>> Sleet 3
```

Mystery Problem #2

15

Class Diagram



```
Snow var2 = new Rain();
var2.method1();
```

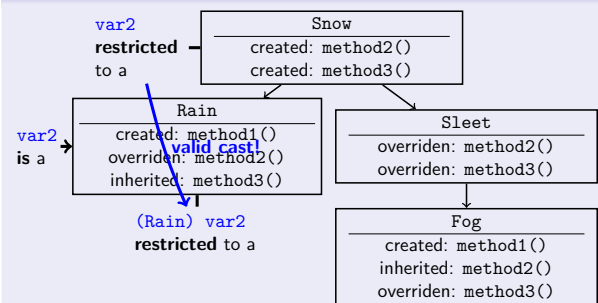
_____ OUTPUT _____

```
>> Rain 2
```

Mystery Problem #3

16

Class Diagram



```
Snow var2 = new Rain();
((Rain) var2).method1();
```

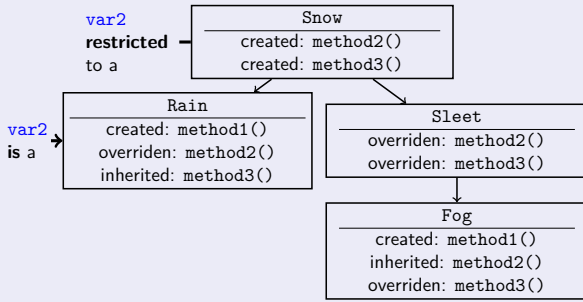
_____ OUTPUT _____

```
>> Rain 1
```

Mystery Problem #4

17

Class Diagram



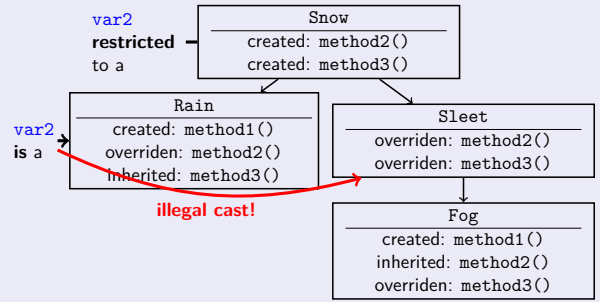
```
Snow var2 = new Rain();
var2.method2();
```

>> Rain 2 OUTPUT

Mystery Problem #5

18

Class Diagram



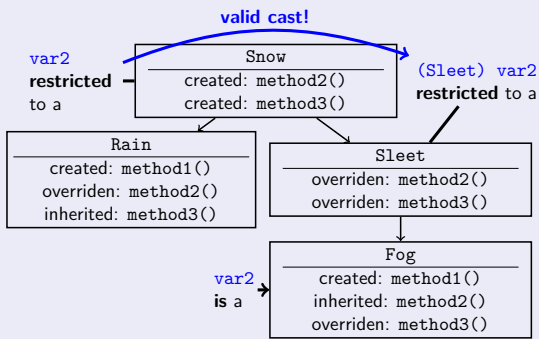
```
Snow var2 = new Rain();
((Sleet) var2).method2();
```

>> ClassCastException: *Error* OUTPUT

Mystery Problem #6

19

Class Diagram



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

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