CS 145

Chapter 9 – Polymorphism

GENERIC OBJECTS

Warm up

Q. Does Java know how to print an object? NO

Q. Can you call the `toString()` on your object without the toString() method implementation?

```
Employee anne = new Employee(2);

System.out.println (anne);
```

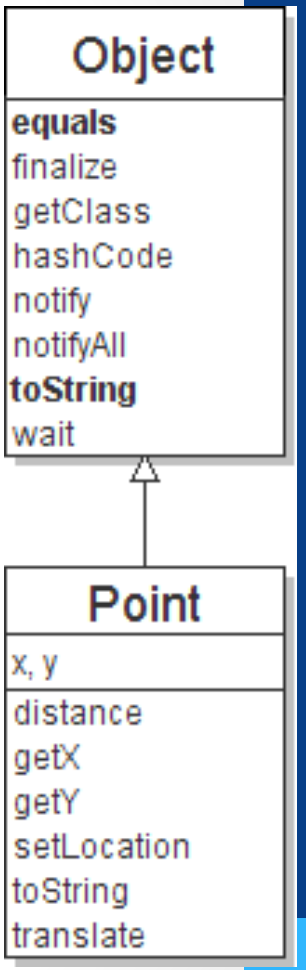
```
Employee@7852e922
```

Class `Object`

- All types of objects have a superclass named `Object`.
  - Every class implicitly extends `Object`

- The `Object` class defines several methods:
  - `public String toString()`
    Returns a `String` representation of the object,
    often so that it can be printed (very useful for debugging)

  - `public boolean equals(Object other)`
    Compare the object to any other for equality.
    Returns `true` if the objects have equal state.



`Object` variables

- You can store any object in a variable of type `Object`.

```
Object o1 = new Point(5, -3);
Object o2 = "hello there";
Object o3 = new Scanner(System.in);
```

- An `Object` variable only knows how to do general things.

```
String s = o1.toString();       // ok
int len = o2.length();          // error
String line = o3.nextLine();    // error
```

- You can write methods that accept an `Object` parameter.

```
public void checkForNull(Object o) {
    if (o == null) {
        throw new IllegalArgumentException();
    }
}
```

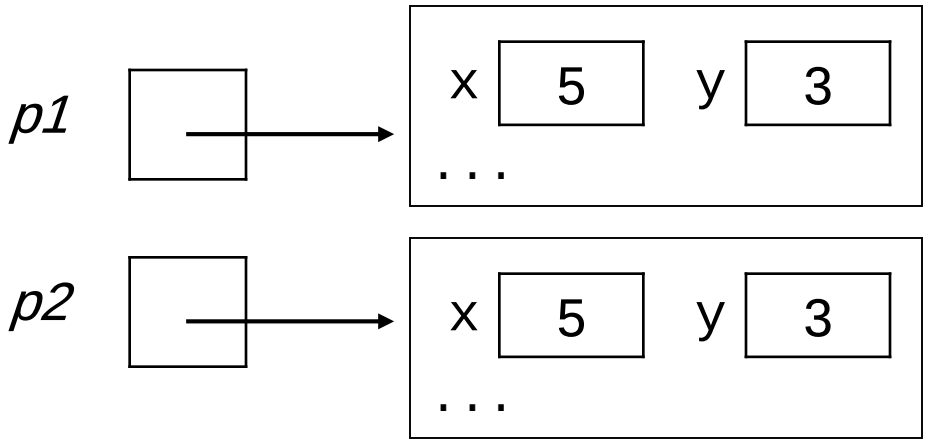
Recall: comparing objects

- The == operator does not work well with objects.

  == compares references to objects, not their state.

  It only produces `true` when you compare an object to itself.

```
Point p1 = new Point(5, 3);
Point p2 = new Point(5, 3);
if (p1 == p2) {   // false
    System.out.println("equal");
}
```



The `equals` method

- The `equals` method compares the state of objects.

```
if (str1.equals(str2)) {
    System.out.println("the strings are equal");
}
```

- But if you write a class, its `equals` method behaves like `==`

```
if (p1.equals(p2)) {   // false :-(
    System.out.println("equal");
}
```

  - This is the behavior we inherit from class `Object`.
  - Java doesn't understand how to compare `Points` by default.

One Version : `equals` method

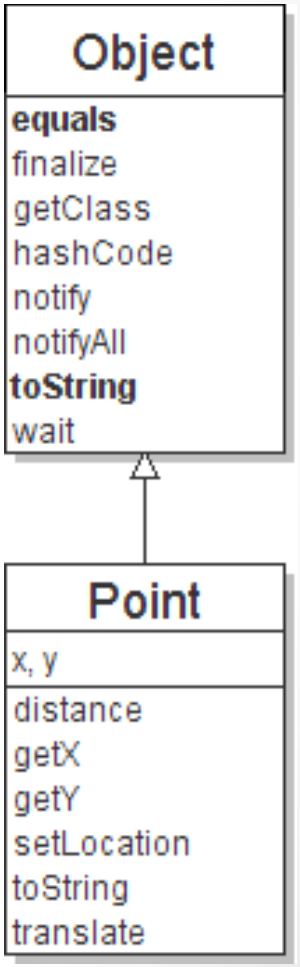
- We can change this behavior by writing an `equals` method.
  - Ours will *override* the default behavior from class `Object`.
  - The method should compare the state of the two objects and return `true` if they have the same x/y position.

- A standard implementation:

```
public boolean equals(Point other) {
 if (x == other.getX() && y == other.getY()) {
        // the objects have equal state
       return true;
    } else {
       return false;
    }
}
```

One example

- Equals is just one of the main types of overridden methods used by most classes.
- To the right you can see the some of the others that are part of object.
- The two most "important" ones
  - equals
  - toString



Review: Polymorphism

- **polymorphism**: Ability for the same code to be used with different types of objects and behave differently with each.
  - `System.out.println` can print any type of object.
    - Each one displays in its own way on the console.

Review: Polymorphism

- You can pass any subtype of a parameter's type.

```
public static void main(String[] args)
   {
       Employee anne   = new Employee();
       Lawyer   bob    = new Lawyer();
       Employee cara   = new Lawyer();

       printEmployee(anne);
       printEmployee(bob);
       printEmployee(cara);

   }

   public static void printEmployee(Employee e)
   {
     System.out.println("This employee works " +
                        e.getHours()+".");
     System.out.println("  And uses the " +
                        e.getVacationForm() + " form");
   }
```

POLYMORPHISM PROBLEMS

Polymorphism problems

- 4-5 classes with inheritance relationships are shown.
- A client program calls methods on objects of each class.
- You must read the code and determine the client's output.
- *<cough> I always put such a question on midterm exams and usually on the final <cough>*

A polymorphism problem

- Suppose that the following four classes have been declared:

```
public class Foo {
    public void method1() {
        System.out.println("foo 1");
    }

    public void method2() {
        System.out.println("foo 2");
    }

    public String toString() {
        return "foo";
    }
}

public class Bar extends Foo {
    public void method2() {
        System.out.println("bar 2");
    }
}
```

A polymorphism problem

```
public class Baz extends Foo {
    public void method1() {
        System.out.println("baz 1");
    }
}
public class Mumble extends Baz {
    public void method2() {
        System.out.println("mumble 2");
    }
}
```

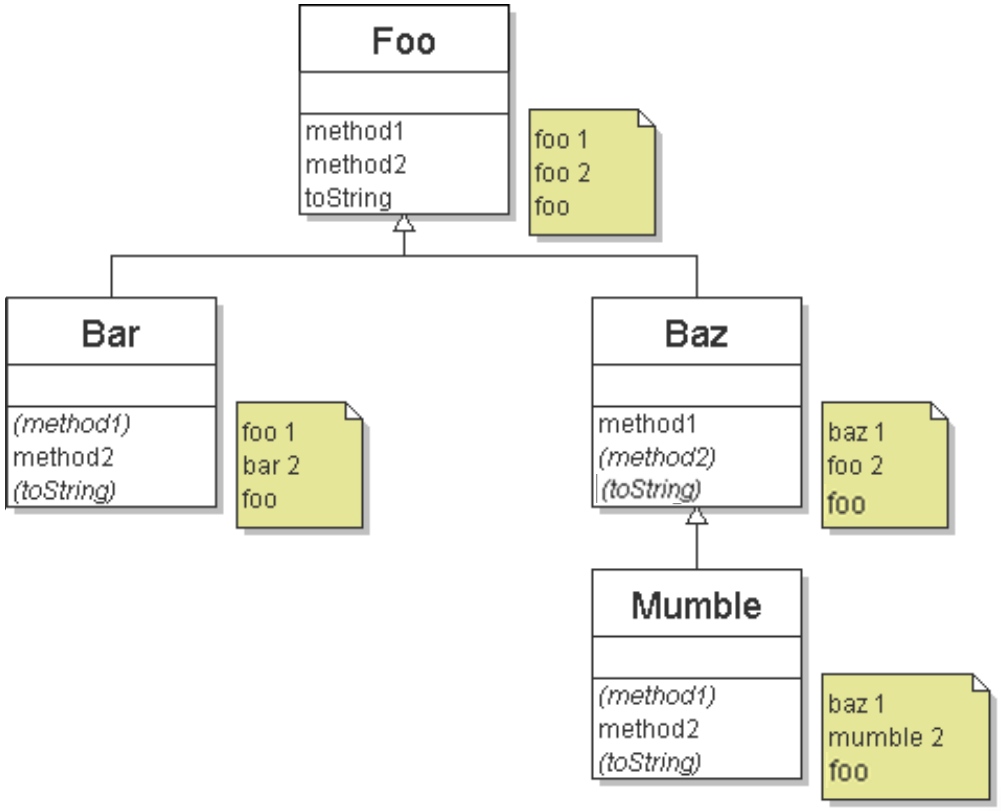
A polymorphism problem

- What would be the output of the following client code?

```
Foo[] pity = {new Baz(), new Bar(), new Mumble(),
  new Foo()};
for (int i = 0; i < pity.length; i++) {
    System.out.println(pity[i]);
    pity[i].method1();
    pity[i].method2();
    System.out.println();
}
```

Diagramming the classes

- Add classes from top (superclass) to bottom (subclass).
- Include all inherited methods.



Polymorphism answer

```
Foo[] pity =   {new Baz(), new Bar(),
                new Mumble(), new Foo()};

for (int i = 0; i < pity.length; i++) {
    System.out.println(pity[i]);
    pity[i].method1();
    pity[i].method2();
    System.out.println();
}
```

- Output:

```
foo
baz 1
foo 2

foo
foo 1
bar 2

foo
baz 1
mumble 2

foo
foo 1
foo 2
```

EXAMPLE #2

Another exercise

- Assume that the following classes have been declared:

```
public class Snow {
    public void method2() {
        System.out.println("Snow 2");
    }

    public void method3() {
        System.out.println("Snow 3");
    }
}

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

    public void method2() {
        System.out.println("Rain 2");
    }
}
```

Exercise

```
public class Sleet extends Snow {
    public void method2() {
        System.out.println("Sleet 2");
        super.method2();
        method3();
    }

    public void method3() {
        System.out.println("Sleet 3");
    }
}

public class Fog extends Sleet {
    public void method1() {
        System.out.println("Fog 1");
    }

    public void method3() {
        System.out.println("Fog 3");
    }
}
```

```
public class Snow {
    public void method2() {
        System.out.println("Snow 2");
    }

    public void method3() {
        System.out.println("Snow 3");
    }
}

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

    public void method2() {
        System.out.println("Rain 2");
    }
}

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

    public void method3() {
        System.out.println("Sleet 3");
    }
}

public class Fog extends Sleet {
    public void method1() {
        System.out.println("Fog 1");
    }

    public void method3() {
        System.out.println("Fog 3");
    }
}
```

What happens when the following examples are executed?

- Example 1:

```
Snow var1 = new Sleet();
var1.method2();
```

- Example 2:

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

- Example 3:

```
Snow var3 = new Fog();
var3.method2();
```

- Example 4:

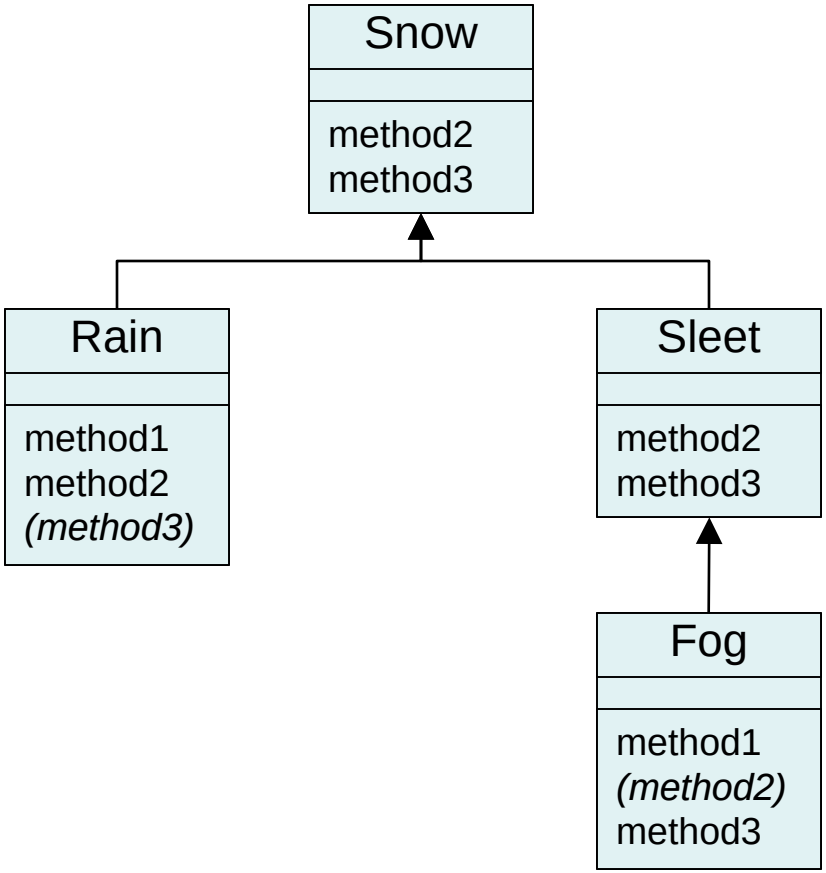
```
Fog var4 = new Snow();
var4.method3();
```

- Example 5:

```
Sleet var5 = new Fog();
var5.method3();
```

Technique 1: diagram

- Diagram the classes from top (superclass) to bottom.



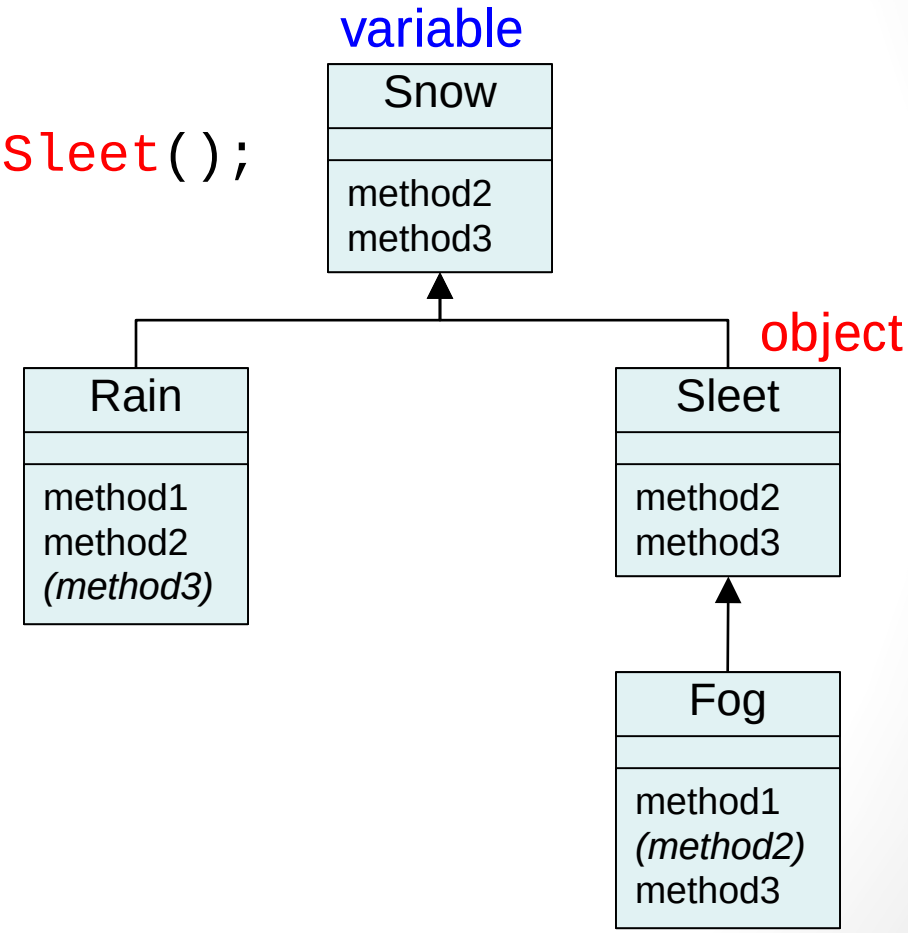
Example 1

- Example:

```
Snow var1 = new Sleet();
var1.method2();
```

- Output:

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



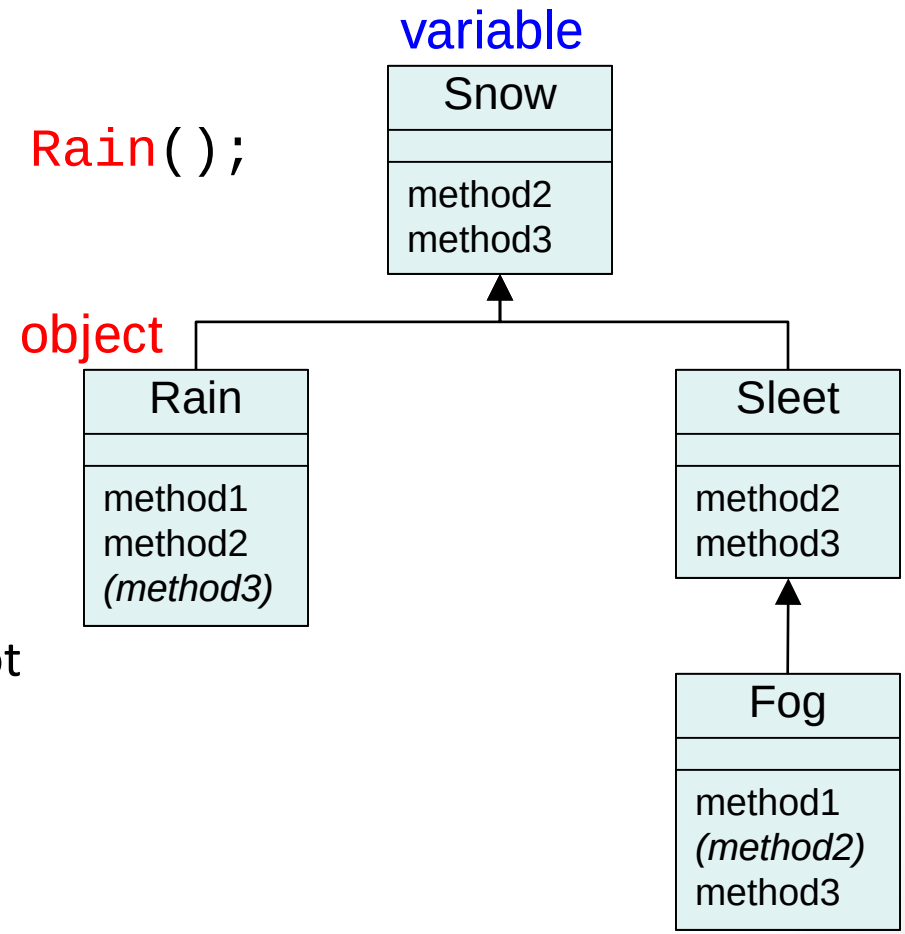
Example 2

- Example:

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

- Output:

None!
There is an error, because `Snow` does not have a `method1`.



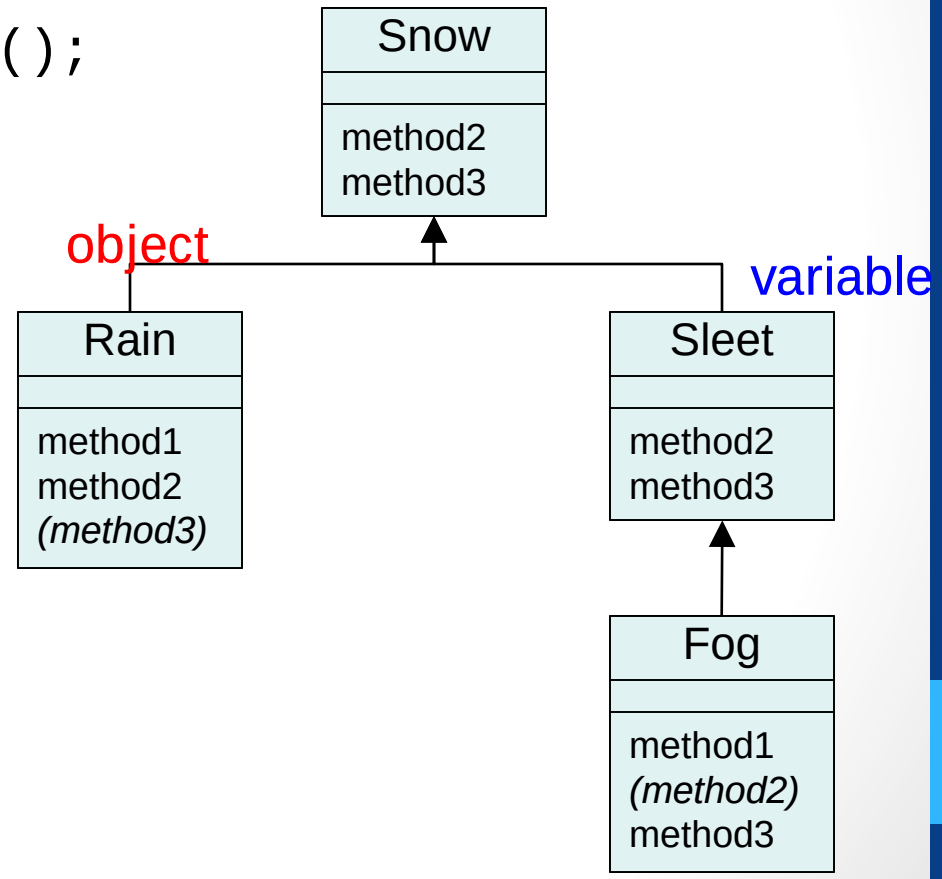
Example 3

- Example:

```
Snow var3 = new Fog();
var3.method2();
```

- Output:

Sleet 2
Snow 2
Fog 3



Example 4

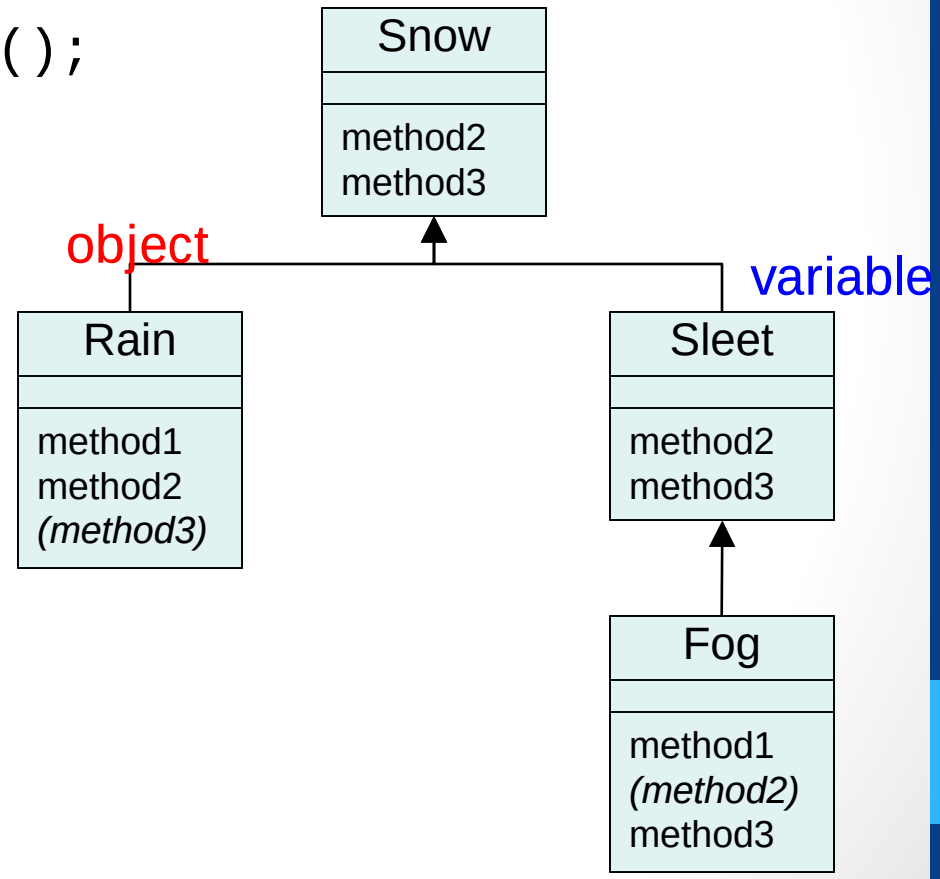
- Example:

```
Fog var4 = new Snow();
var4.method3();
```

- Output:

None!

There is an error, because Snow does not have fit into a Fog box.



Example 5

- Example:

```
Sleet var5 = new Fog();
var5.method3();
```

- Output:

Fog 3