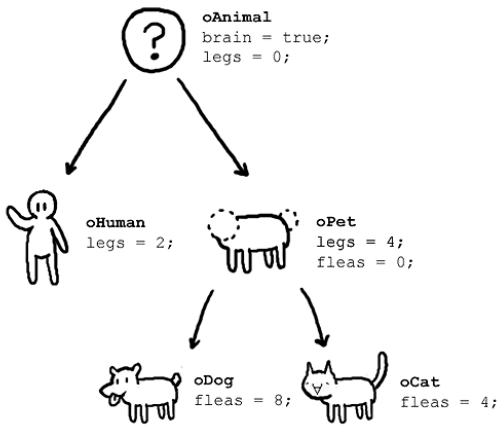
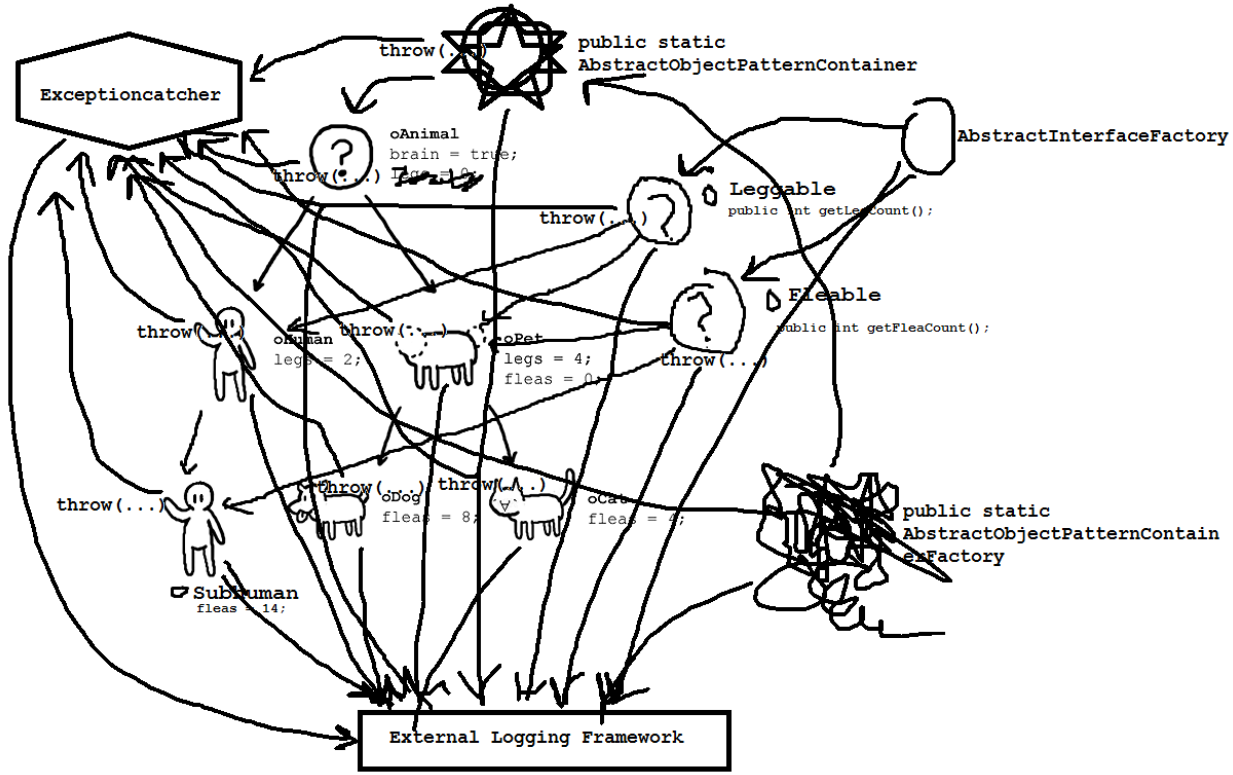
What OOP users claim



What actually happens



CS 2112 Lab: Prelim Review

September 30 / October 2, 2019

Inheritance Overview

- Language mechanism for extending and reusing code
- Different from subtyping!
- Two basic functions: Copying and Editing

Copying and Editing

- Copying is provided by the keyword `extends` in the method header
- This allows you to use any functionality you included in your superclass, as long as it is public (or protected)
- You can edit existing classes by adding or changing functionality in a subclass
- Any time you extend a class, you create a subtyping relationship where subclass $<:$ superclass

An Example

```
1  class Robot {
2    ...
3
4    public void doSomething() { ... }
5  }
```

```
1  class SmartRobot extends Robot {
2        ...
3    private int numSomethingsDone;
4
5    public void doSomething() {
6      ...
7      numSomethingsDone++;
8    }
9  }
```

Method Dispatch

```
1    Robot roboMan = new SmartRobot();
2
3    roboMan.doSomething();
```

Which doSomething() is called?

Method Dispatch

- The static type is `Robot` and the dynamic type is `SmartRobot`
- This method is not static, so the method `doSomething()` of the dynamic type is called
- After this call, `numSomethingsDone` $= 1$

Method Dispatch

```
1  class Robot {
2        ...
3    public void doSomething() { ... }
4
5    public void doSomethingElse() {
6      doSomething();
7    }
8  }
```

```
1  Robot roboMan = new SmartRobot();
2
3  roboMan.doSomethingElse();
```

Now, which doSomething() is called?

Method Dispatch

- Even if this call is made within a method of the superclass, the `doSomething()` method in the subclass will still be called
- This is called *late binding*

Static Methods

```
1     public Robot {
2       static String hello() {
3         return "HELLO";
4       }
5     }
6     public SmartRobot extends Robot {
7       static String hello() {
8         return "Hello!";
9       }
10    }
```

```
1     Robot roboMan = new SmartRobot();
2     roboMan.hello();
```

What is returned?

Instance Variables

There are some rare cases where the "copied down" view is not quite accurate. For example, a method in the superclass can refer to a field in the superclass that is shadowed by a field with the same name in a subclass. If the method in the superclass refers to this field, then it still refers to the same field even after it is copied down to the subclass.

Static Methods

- The `hello()` method in the static type would be called
- That method would return "HELLO"

Static Methods

Which will work?

```
1    Robot roboman = new Robot();
2    Robot.hello();
```

```
1    Robot roboman;
2    roboman.hello();
```

```
1    Robot roboman = null;
2    roboman.hello();
```

Constructors

- To make sure you don't leave anything uninitialized, Java requires that you call the superclass constructor in the first line of your subclass constructor
- If you don't, Java will call `super()` automatically

Protected Visibility

- Visibility modifier `protected` will be accessible to the class and any of its subclasses
- This creates a specialization interface that allows others to edit and expand your code without changing the public interface
- Public and protected methods can be overridden, while private ones cannot
- This is why it is good practice to create a specialization interface – you can define the way in which your code can be extended

Review