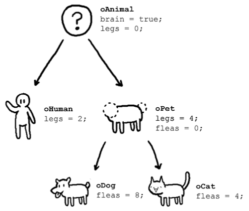
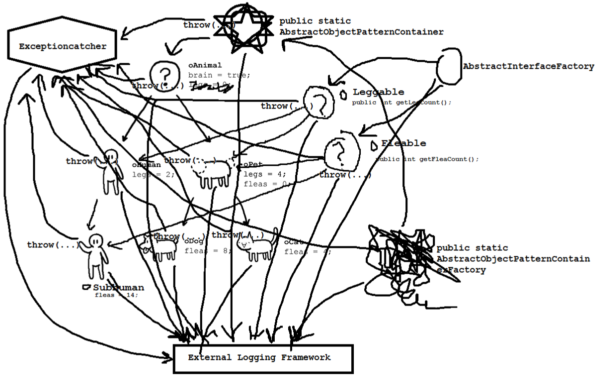
What OOP users claim



What actually happens



Lab 6: Prelim Review

CS 2112 Fall 2020

October 19 / 21, 2020

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

```
class Robot {
  ...

  public void doSomething() { ... }
}
```

```
class SmartRobot extends Robot {
      ...
  private int numSomethingsDone;

  public void doSomething() {
    ...
    numSomethingsDone++;
  }
}
```

Method Dispatch

```java
Robot roboMan = new SmartRobot();

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

```
class Robot {
      ...
  public void doSomething() { ... }

  public void doSomethingElse() {
    doSomething();
  }
}
```

```
Robot roboMan = new SmartRobot();

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

```
public Robot {
  static String hello() {
    return "HELLO";
  }
}
public SmartRobot extends Robot {
  static String hello() {
    return "Hello!";
  }
}
```

```
Robot roboMan = new SmartRobot();
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?

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

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

```
    Robot roboman = null;
    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