Software Development (CS2500)

Lecture 2: Introduction to Objects

M. R. C. van Dongen

September 25, 2013

Chair Wars

A Play in Four Acts

- Act I: The Challenge.
- Act II: Changing the Specs.
- Act III: They both Got it Wrong.
- Act IV: And the Winner is ....

Prelude

- Mr Billy owns a *huge* computer company.
- Software Development is the most profitable division.
- Larry has been working there for years.
- Brad has only just started.
- One day, Mr Billy calls them in to his office.

Act I: The Challenge

Act I: The Challenge

Act I: The Challenge

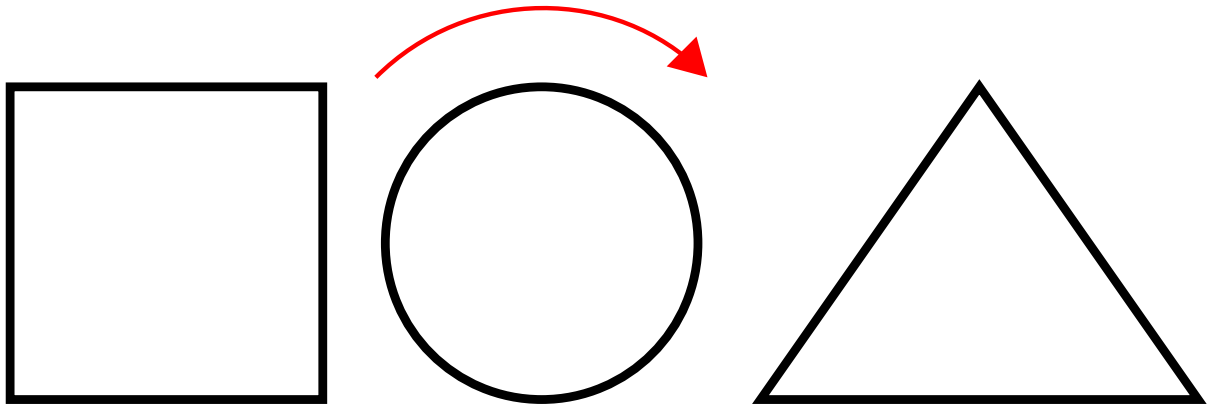
Act I: The Challenge

Act I: The Challenge

The Spec

- There will be shapes on a GUI:
  - A square;
  - A circle; and
  - A triangle.
- When the user clicks on the shape:
  - The shape should rotate 360° clockwise.
  - The app should play an `aif` sound.
  - The sound is specific to the shape.

The Spec



The Prize

The Prize

The Prize: An Aeron

The Prize: An Aeron

The Prize: An Aeron

Introducing the Contestants: Meet Larry

- Procedural Programmer.
- Always thinking about what his programs should do.
- Always on the lookout for procedures.

Introducing the Contestants: Meet Brad

- Object Oriented Programmer.
- Always thinking about the things in his program.
- Always on the lookout for players and objects.

Meanwhile in Larry's Cubicle

- Immediately sets about *writing* his important procedures.
- He wrote `rotate( )` and `playSound( )` in no time.

Meanwhile in Larry's Cubicle

- Immediately sets about *writing* his important procedures.
- He wrote `rotate( )` and `playSound( )` in no time.

Procedural Programming Language

```
rotate( shapeNum ) { /* Make shape rotate 360 degrees. */ }
playSound( shapeNum ) { /* Look up sound and play it. */ }
```

Meanwhile at Brad's Laptop at the Cafe

- Sets about *selecting* his important *players*:
  - The objects: the shapes.
- He wrote a separate *class* for each of the three shapes.

Meanwhile at Brad's Laptop at the Cafe

- Sets about *selecting* his important *players*:
  - The objects: the shapes.
- He wrote a separate *class* for each of the three shapes.

Object Oriented Programming Language

```
public class Square {
  rotate( ) { /* Make square rotate 360 degrees. */ }
  playSound( ) { /* Play square's sound. */ }
}
```

Meanwhile at Brad's Laptop at the Cafe

- Sets about *selecting* his important *players*:
  - The objects: the shapes.
- He wrote a separate *class* for each of the three shapes.

Object Oriented Programming Language

```
public class Circle {
  rotate( ) { /* Make circle rotate 360 degrees. */ }
  playSound( ) { /* Play circle's sound. */ }
}
```

Meanwhile at Brad's Laptop at the Cafe

- Sets about *selecting* his important *players*:
  - The objects: the shapes.
- He wrote a separate *class* for each of the three shapes.

Object Oriented Programming Language

```
public class Triangle {
  rotate( ) { /* Make triangle rotate 360 degrees. */ }
  playSound( ) { /* Play triangle's sound. */ }
}
```

Meanwhile in Larry's Cubicle

- Quickly tested his application.
- Knew he had beaten Brad.
- He was about to claim his chair.

Meanwhile in Larry's Cubicle

- Quickly tested his application.
- Knew he had beaten Brad.
- He was about to claim his chair.
- But then

Meanwhile in Larry's Cubicle

- Quickly tested his application.
- Knew he had beaten Brad.
- He was about to claim his chair.
- But then, all of a sudden

Meanwhile in Larry's Cubicle

- Quickly tested his application.
- Knew he had beaten Brad.
- He was about to claim his chair.
- But then, all of a sudden, there was a change in the specs.

Act II: Changing the Specs

Sorry guys. I've been on the phone with the clients.

Act II: Changing the Specs

If we can't get them an amoeba shape they're gonna take their business elsewhere.

Act II: Changing the Specs

Act II: Changing the Specs

The amoeba has a special `.hif` sound file.

Meanwhile in Larry's Cubicle

- His `rotate( )` would still work.
- But his `playsound( )` would have to change.
- *Changing existing code* doesn't make him feel too good.

Meanwhile in Larry's Cubicle

- His `rotate( )` would still work.
- But his `playsound( )` would have to change.
- *Changing existing code* doesn't make him feel too good.

Procedural Programming Language

```
playSound( shapeNum ) {
  // if shape is not amoeba
      // lookup sound file and play as .aif
  // else
      // play amoeba .hif sound }
```

Meanwhile at Brad's Laptop

- Smiles all over the place.
- All he has to do is write a *new* class for amoeba objects.
- *No need to change* any existing code.

Meanwhile at Brad's Laptop

- Smiles all over the place.
- All he has to do is write a *new* class for amoeba objects.
- *No need to change* any existing code.

Object Oriented Programming Language

```
public class Amoeba {
  rotate( ) { /* Make amoeba rotate 360 degrees. */ }
  playSound( ) { /* Play amoeba's .hif sound. */ }
}
```

Meanwhile in Larry's Cubicle

- Has to test `playsound( )` for all shapes.
- Has to test `rotate( )` for amoeba.

Meanwhile at Brad's Laptop

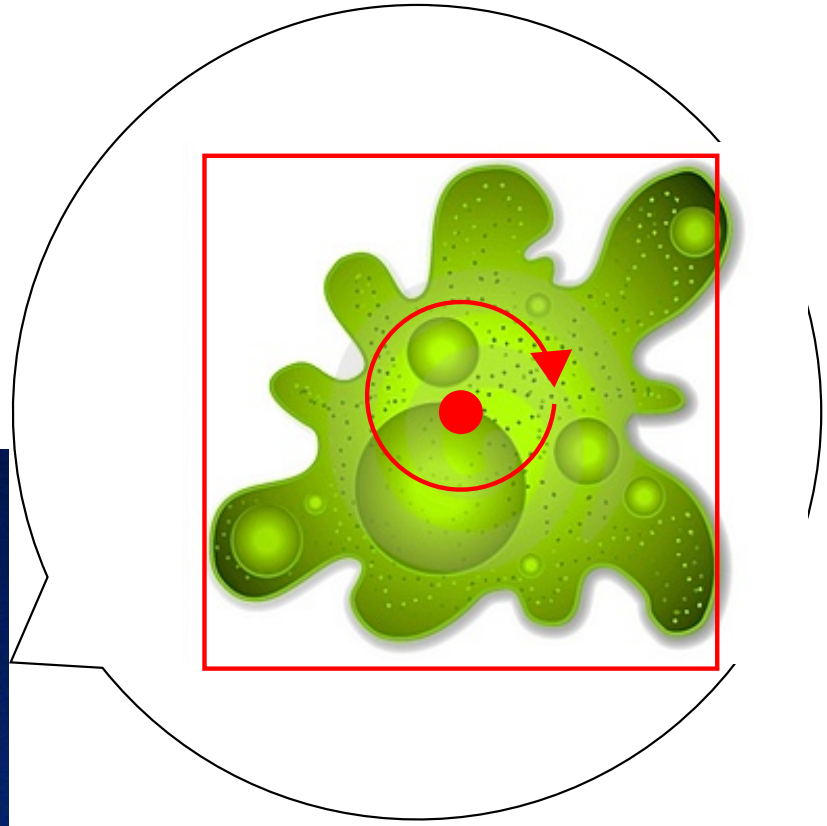
- Hasn't changed any code.
- Only has to test the new amoeba shape.

Act III: They Both got it Wrong

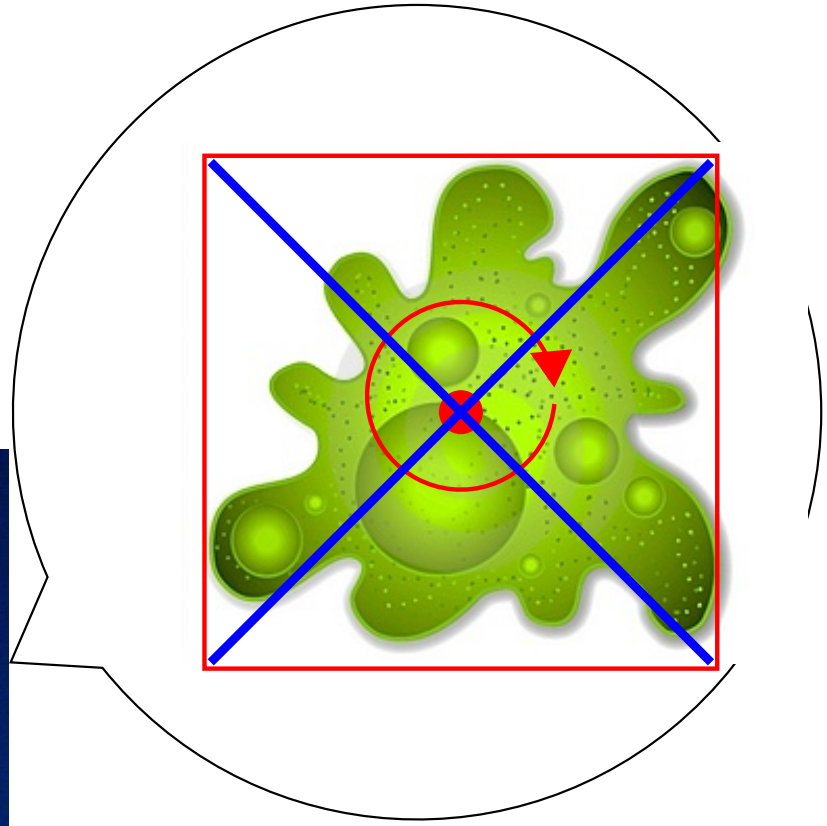
Act III: They Both got it Wrong

That's not how an amoeba is supposed to rotate.

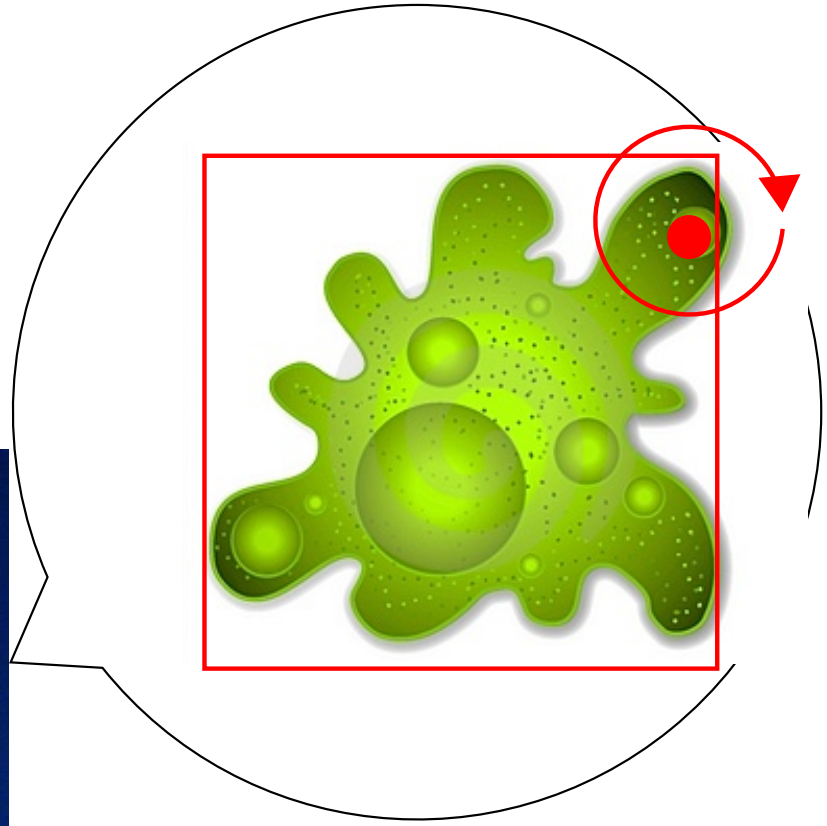
Act III: They Both got it Wrong



Act III: They Both got it Wrong



Act III: They Both got it Wrong



Meanwhile in Larry's Cubicle

- Can't stand it: he had almost won the chair.
- Has to change the *entire* `rotate( )` procedure.
- Has to test `rotate( )` for *all* shapes.

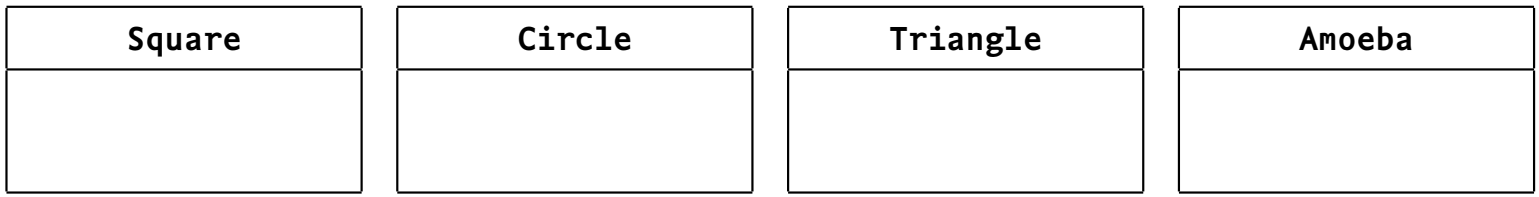
Meanwhile at Brad's Laptop

- Smiled.
- *Only* has to change the `rotate( )` method in the `Amoeba` class.
- *Only* has to test the `rotate( )` method in the `Amoeba` class.

Final Act: And the Winner Is ...

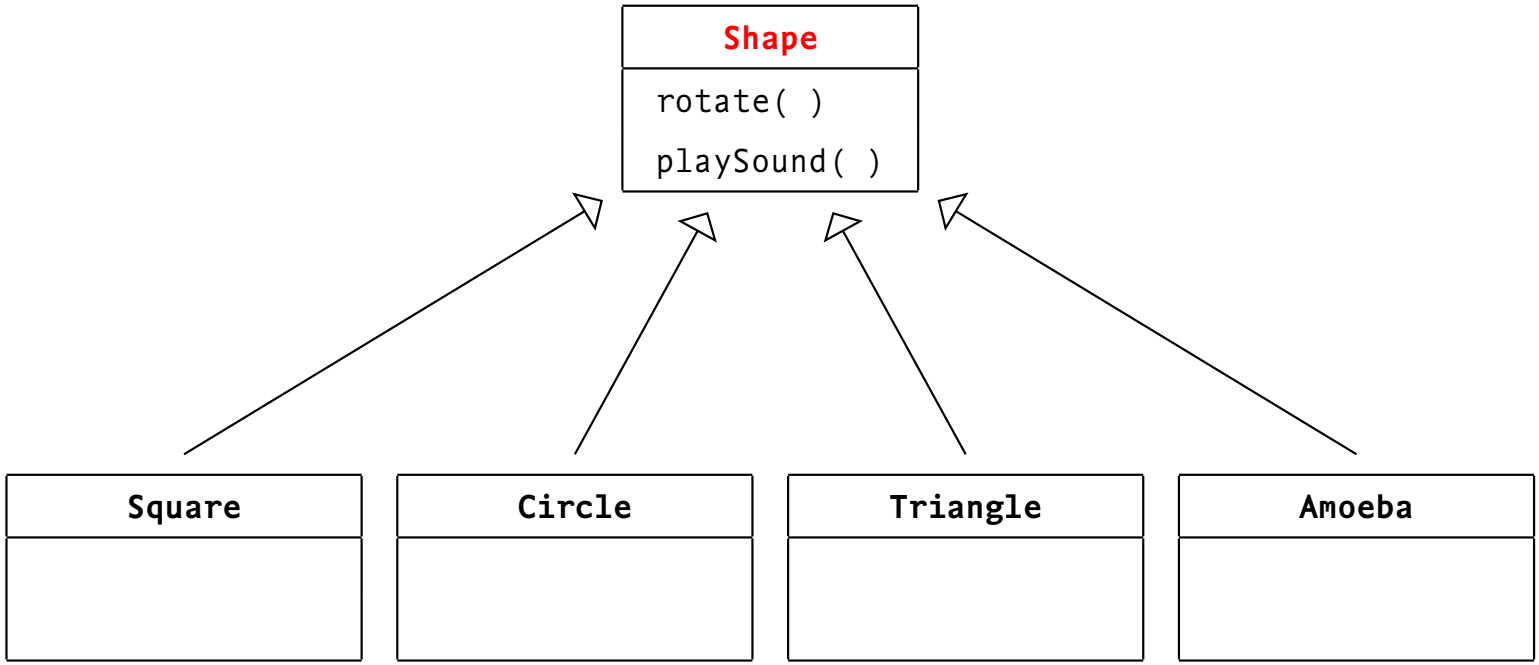
- Brad has beaten Larry to it.
- But Larry thinks he has spotted a weakness:
  - He thinks Brad has lots of duplicated code.

Final Act: And the Winner Is ...



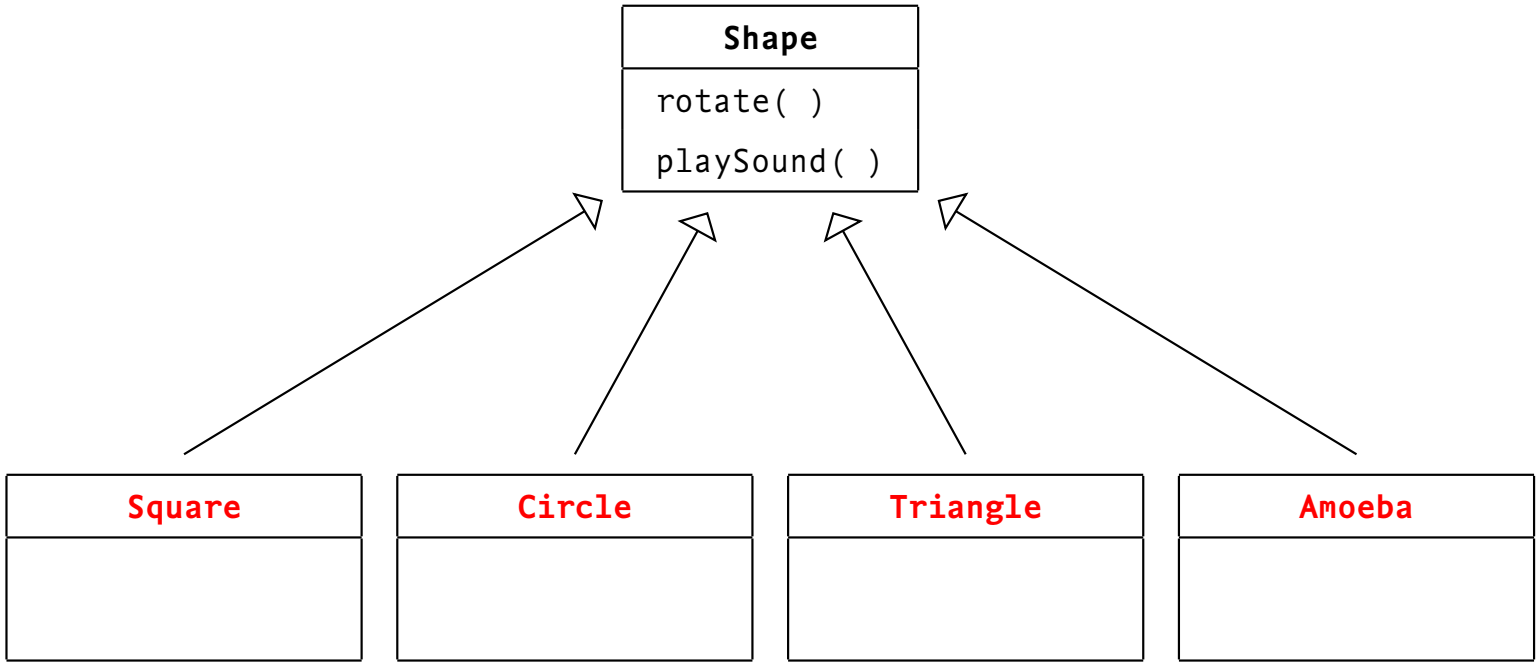
Final Act: And the Winner Is ...

Superclass



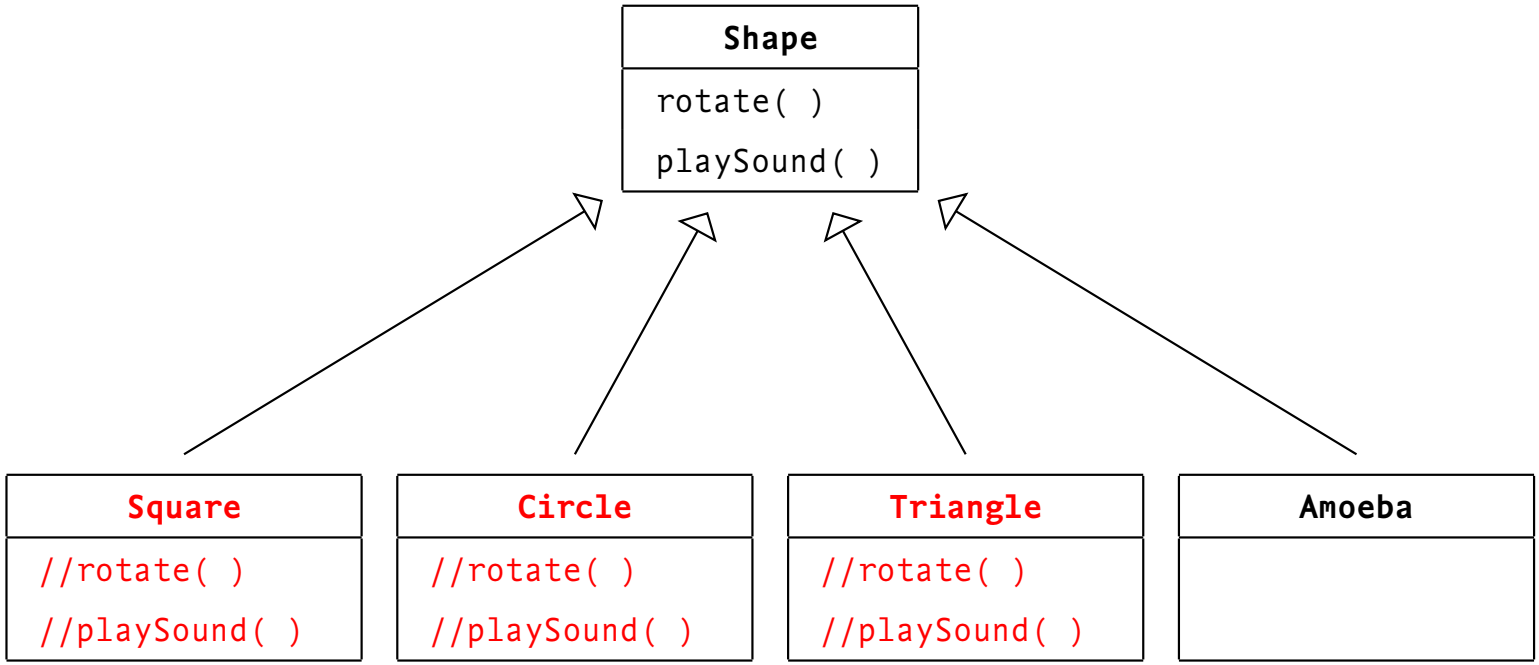
Final Act: And the Winner Is ...

Subclasses



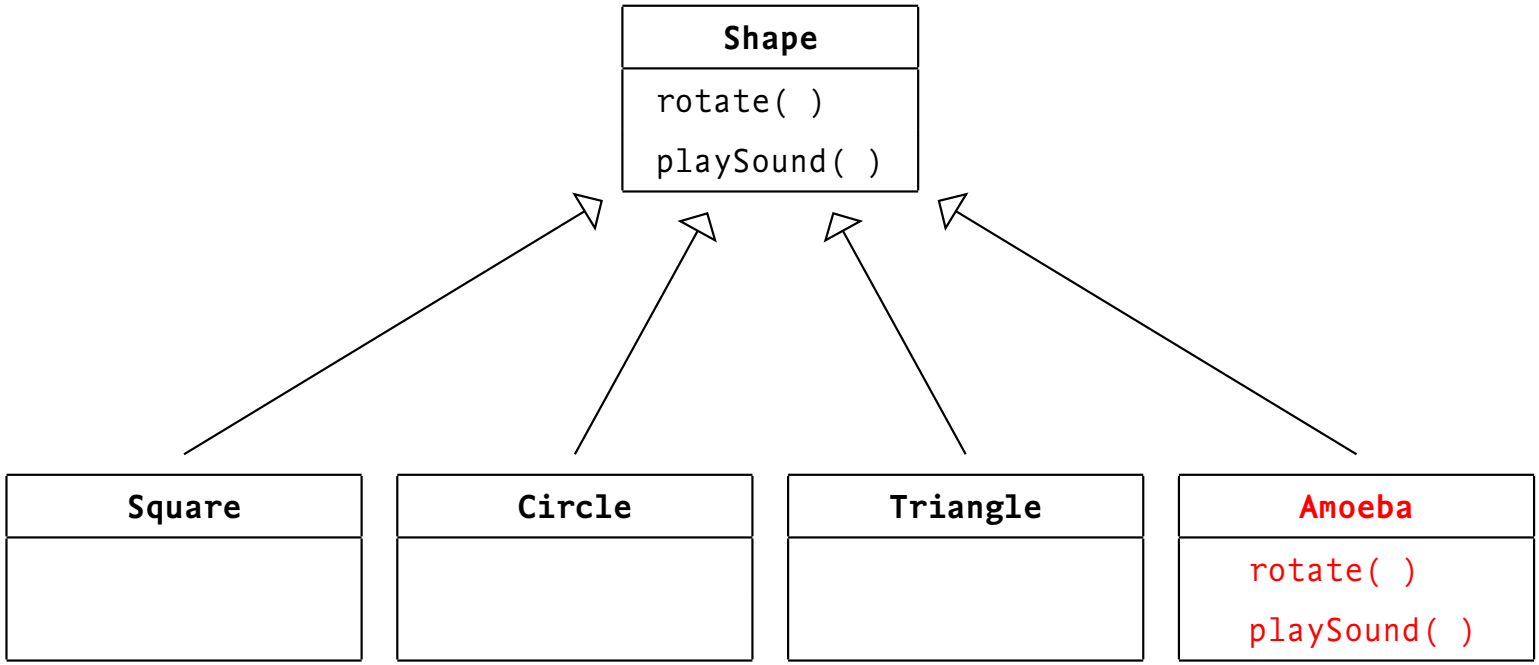
Final Act: And the Winner Is ...

Inherit



Final Act: And the Winner Is ...

Overrides



Final Act: And the Winner Is ...

Sorry Brad. You *were* first.

Final Act: And the Winner Is ...

And you would have won the chair, but ....

Final Act: And the Winner Is ...

I also asked Amy from the 2nd floor to work on the app.

Final Act: And the Winner Is ...

She finished at the same time as you.

Final Act: And the Winner Is ...

Final Act: And the Winner Is ...

But there can only be one winner.

Final Act: And the Winner Is ...

Final Act: And the Winner Is ...

Amy

Final Act: And the Winner Is ...

Amy

State and Behaviour

- Classes are used to *define* your objects.
- Each class corresponds to an *object type*.
- When you create a class, you think about:

  Knowing: What the object knows.

  Doing: What the object does.

State and Behaviour

- Classes are used to *define* your objects.
- Each class corresponds to an *object type*.
- When you create a class, you think about:

  Knowing: What the object knows.
  - The object's *instance variables* define its memory.

  Doing: What the object does.

State and Behaviour

- Classes are used to *define* your objects.
- Each class corresponds to an *object type*.
- When you create a class, you think about:

  Knowing: What the object knows.
  - The object's *instance variables* define its memory.
  - Instance variables are also known as *attributes*.

  Doing: What the object does.

State and Behaviour

- Classes are used to *define* your objects.
- Each class corresponds to an *object type*.
- When you create a class, you think about:

  Knowing: What the object knows.
  - The object's *instance variables* define its memory.
  - Instance variables are also known as *attributes*.
  - The object's attributes define the object's *state*.

  Doing: What the object does.

State and Behaviour

- Classes are used to *define* your objects.
- Each class corresponds to an *object type*.
- When you create a class, you think about:

  Knowing: What the object knows.
  - The object's *instance variables* define its memory.
  - Instance variables are also known as *attributes*.
  - The object's attributes define the object's *state*.

  Doing: What the object does.
  - What object do are its (instance) *methods*.

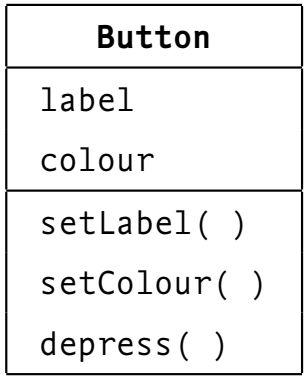
State and Behaviour

- Classes are used to *define* your objects.
- Each class corresponds to an *object type*.
- When you create a class, you think about:

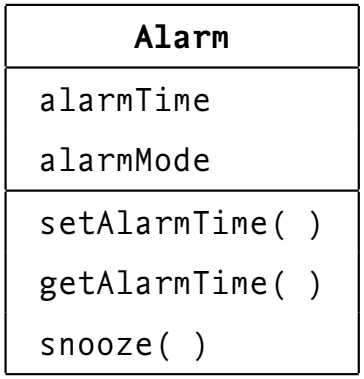
  Knowing: What the object knows.
  - The object's *instance variables* define its memory.
  - Instance variables are also known as *attributes*.
  - The object's attributes define the object's *state*.

  Doing: What the object does.
  - What object do are its (instance) *methods*.
  - The object's instance methods define its *behaviour*.

State and Behaviour



State and Behaviour



How Does it Work in Java?

Java

```
public class Song {
    private String title;  // Attribute: knowing.
    private String artist; // Attribute: knowing.

    public String getTitle( ) { // Method: doing.
        return title;
    }

    public void playSong( ) {   // Method: doing.
        // play song.
    }
    ⋮
}
```

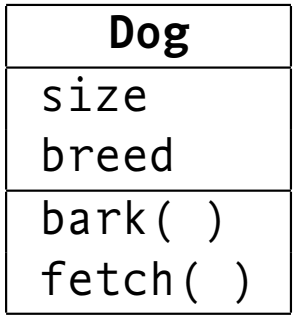
Class versus Object

- A class is not an object.
- A class is used to construct objects at run time.
- The class acts as a *blueprint* for the object.
- One class may define several objects.

Class versus Object

One Class

- A class is not an object.
- A class is used to construct objects at run time.
- The class acts as a *blueprint* for the object.
- One class may define several objects.

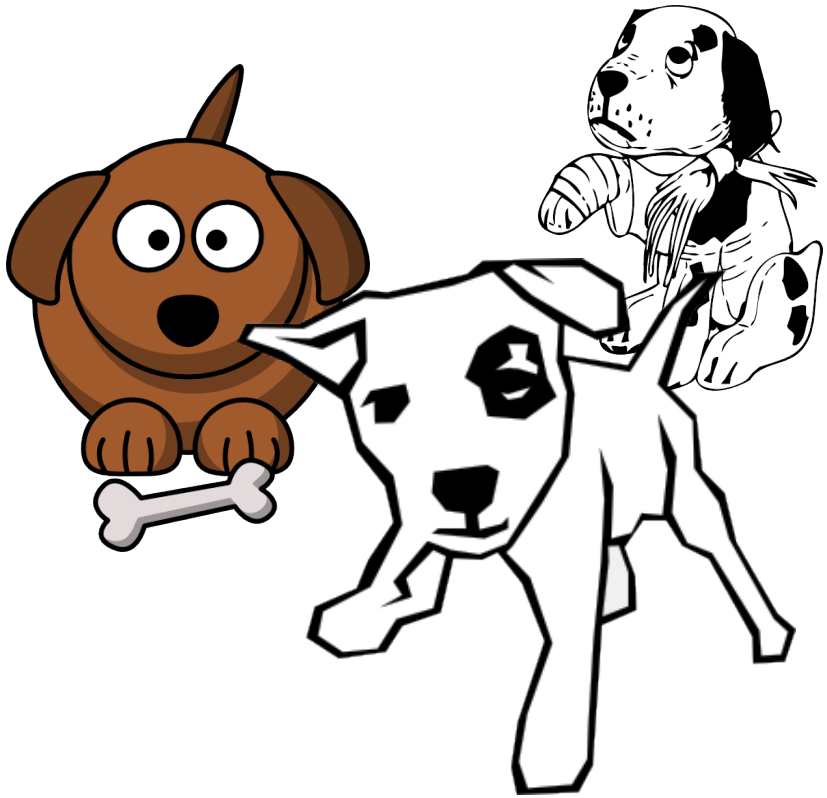


Class versus Object

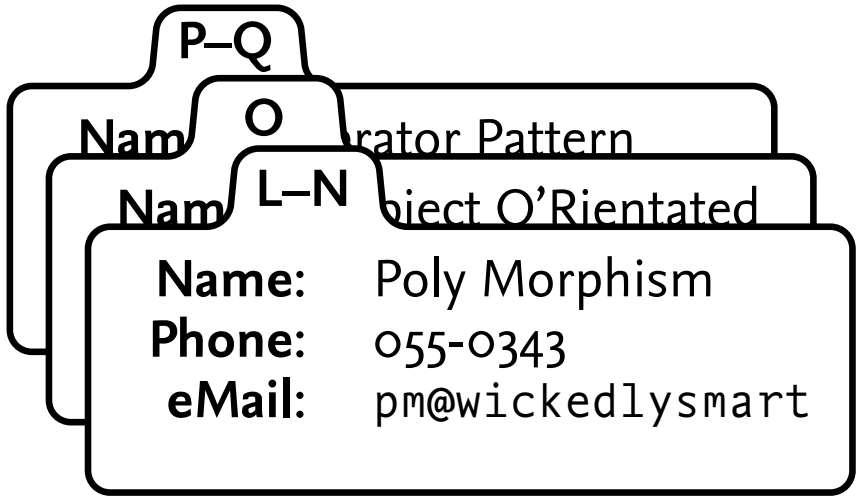
Many Objects

- A class is not an object.
- A class is used to construct objects at run time.
- The class acts as a *blueprint* for the object.
- One class may define several objects.

Dog
`size` `breed`
`bark( )` `fetch( )`

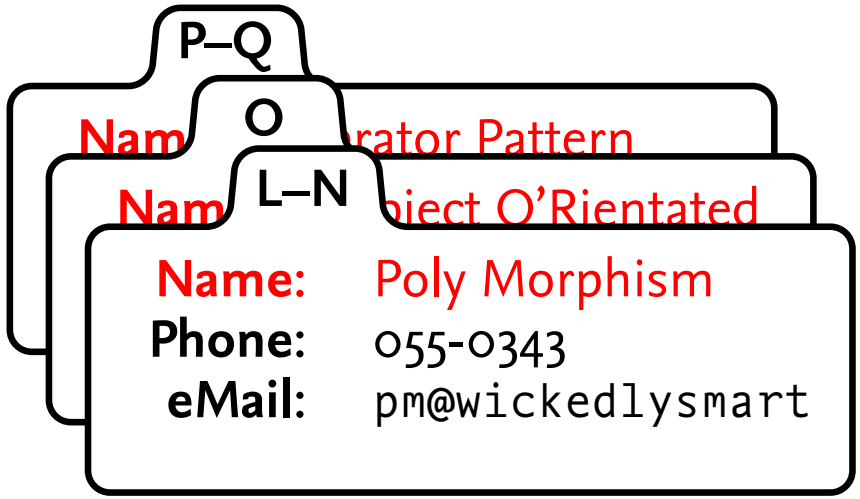


Objects are like Rolodex Cards



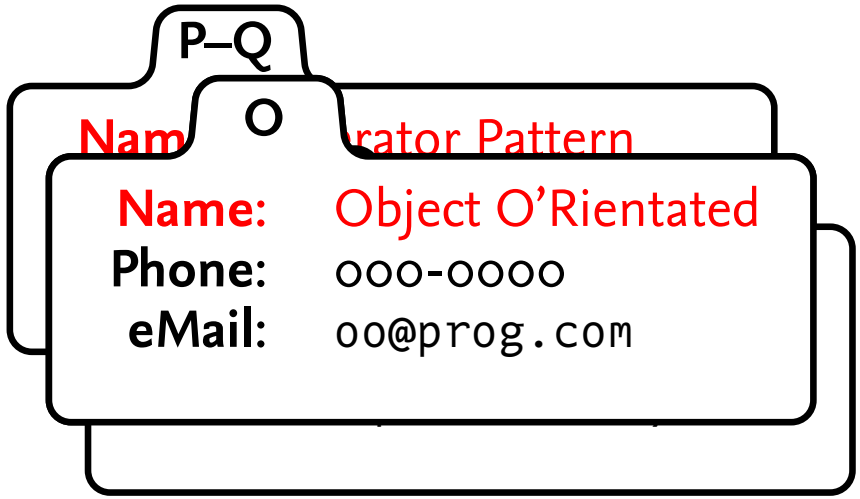
Objects are like Rolodex Cards

Name



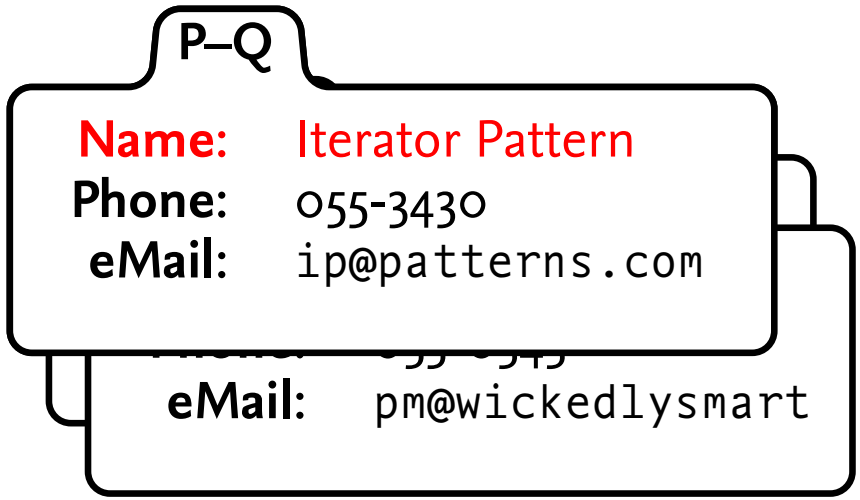
Objects are like Rolodex Cards

Name



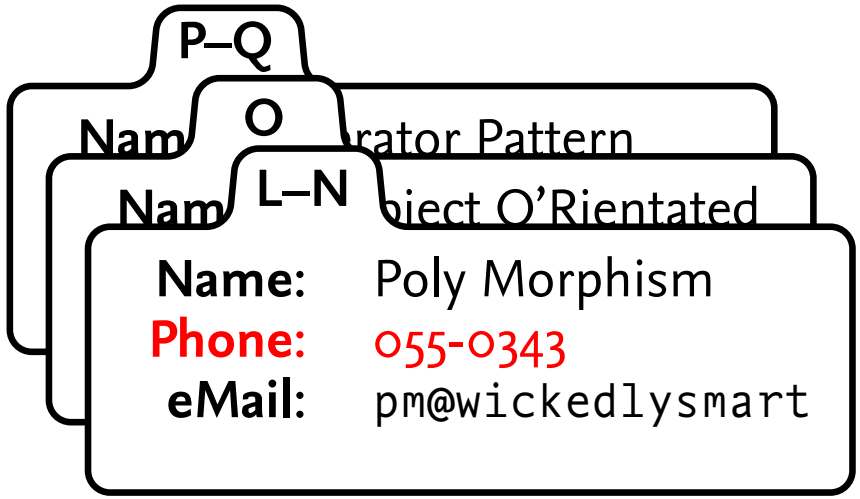
Objects are like Rolodex Cards

Name



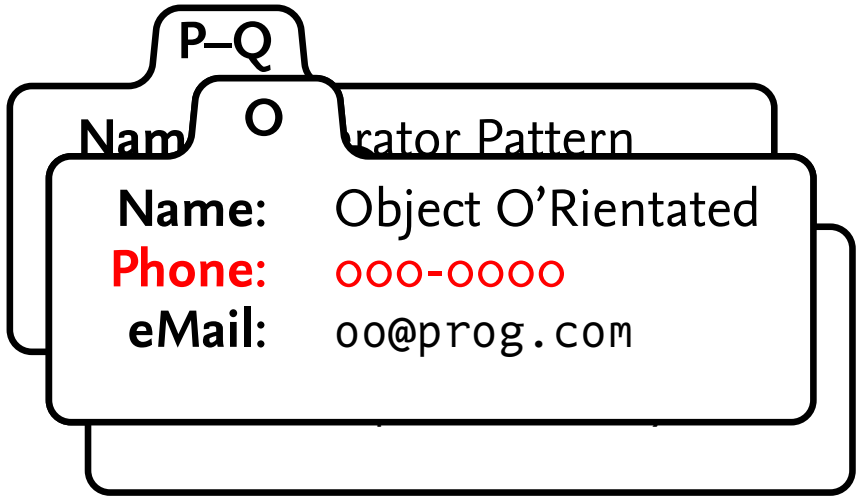
Objects are like Rolodex Cards

Phone



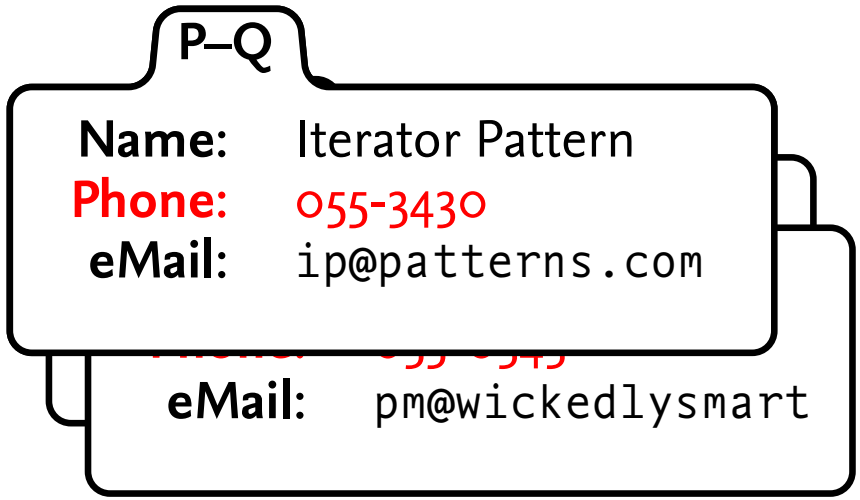
Objects are like Rolodex Cards

Phone



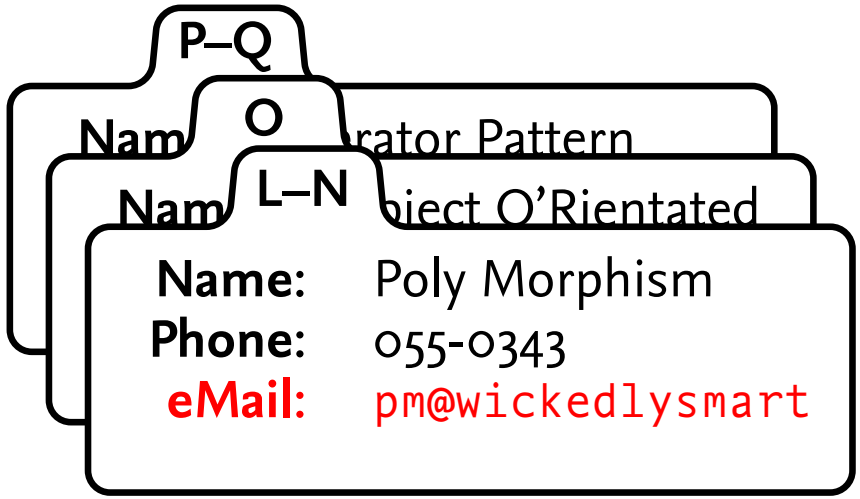
Objects are like Rolodex Cards

Phone



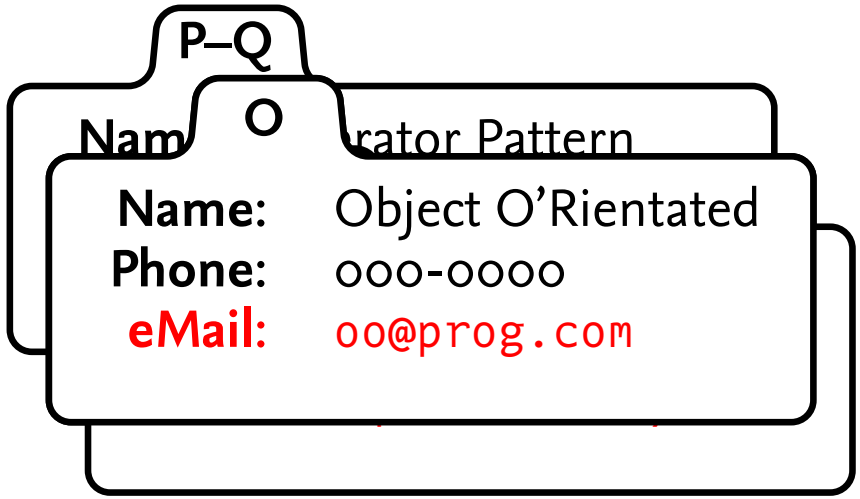
Objects are like Rolodex Cards

eMail



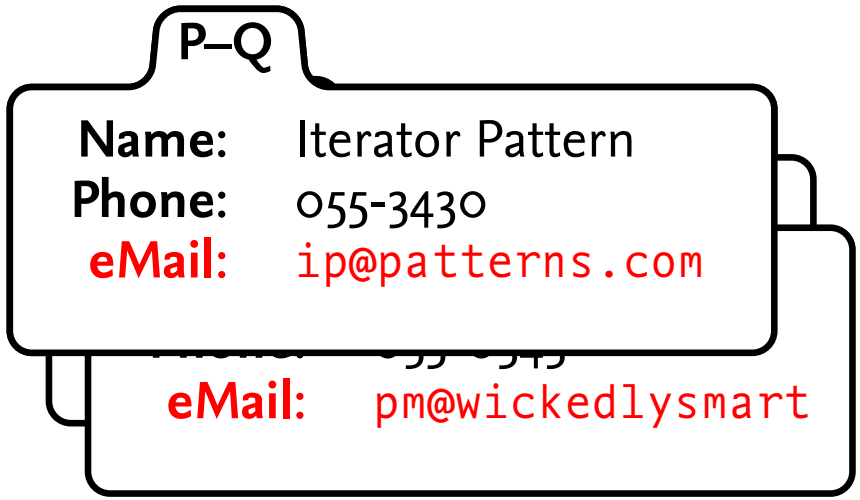
Objects are like Rolodex Cards

eMail



Objects are like Rolodex Cards

eMail



How Does it Work in Java?

Java

```
public class Dog {
  public String breed;
  public double size;


  public void bark( ) {
    if (size > 1.0) {
      System.out.println( "Ruff" );
    } else {
      System.out.println( "bark" );
    }
  }
}
```

Java

```
public class Main {
  public static void main( String[] args ) {
    Dog john = new Dog( );
    john.breed = "Bulldog";
    john.size = 2.0;
    Dog lucky = new Dog( );
    lucky.breed = "Terrier";
    lucky.size = 0.5;

    john.bark( ); // Ruff
    lucky.bark( ); // bark
  }
}
```

How Does it Work in Java?

Create new Dog

Java

```
public class Dog {
  public String breed;
  public double size;


  public void bark( ) {
    if (size > 1.0) {
      System.out.println( "Ruff" );
    } else {
      System.out.println( "bark" );
    }
  }
}
```

Java

```
public class Main {
  public static void main( String[] args ) {
    Dog john = new Dog( );
    john.breed = "Bulldog";
    john.size = 2.0;
    Dog lucky = new Dog( );
    lucky.breed = "Terrier";
    lucky.size = 0.5;

    john.bark( ); // Ruff
    lucky.bark( ); // bark
  }
}
```

How Does it Work in Java?

Set John's Breed

Java

```
public class Dog {
  public String breed;
  public double size;


  public void bark( ) {
    if (size > 1.0) {
      System.out.println( "Ruff" );
    } else {
      System.out.println( "bark" );
    }
  }
}
```

Java

```
public class Main {
  public static void main( String[] args ) {
    Dog john = new Dog( );
    john.breed = "Bulldog";
    john.size = 2.0;
    Dog lucky = new Dog( );
    lucky.breed = "Terrier";
    lucky.size = 0.5;

    john.bark( ); // Ruff
    lucky.bark( ); // bark
  }
}
```

How Does it Work in Java?

Set John's Size

Java

```
public class Dog {
  public String breed;
  public double size;


  public void bark( ) {
    if (size > 1.0) {
      System.out.println( "Ruff" );
    } else {
      System.out.println( "bark" );
    }
  }
}
```

Java

```
public class Main {
  public static void main( String[] args ) {
    Dog john = new Dog( );
    john.breed = "Bulldog";
    john.size = 2.0;
    Dog lucky = new Dog( );
    lucky.breed = "Terrier";
    lucky.size = 0.5;

    john.bark( ); // Ruff
    lucky.bark( ); // bark
  }
}
```

How Does it Work in Java?

Create new Dog

Java

```
public class Dog {
  public String breed;
  public double size;


  public void bark( ) {
    if (size > 1.0) {
      System.out.println( "Ruff" );
    } else {
      System.out.println( "bark" );
    }
  }
}
```

Java

```
public class Main {
  public static void main( String[] args ) {
    Dog john = new Dog( );
    john.breed = "Bulldog";
    john.size = 2.0;
    Dog lucky = new Dog( );
    lucky.breed = "Terrier";
    lucky.size = 0.5;

    john.bark( ); // Ruff
    lucky.bark( ); // bark
  }
}
```

How Does it Work in Java?

Set Lucky's Breed

Java

```
public class Dog {
  public String breed;
  public double size;


  public void bark( ) {
    if (size > 1.0) {
      System.out.println( "Ruff" );
    } else {
      System.out.println( "bark" );
    }
  }
}
```

Java

```
public class Main {
  public static void main( String[] args ) {
    Dog john = new Dog( );
    john.breed = "Bulldog";
    john.size = 2.0;
    Dog lucky = new Dog( );
    lucky.breed = "Terrier";
    lucky.size = 0.5;

    john.bark( ); // Ruff
    lucky.bark( ); // bark
  }
}
```

How Does it Work in Java?

Set Lucky's Size

Java

```
public class Dog {
  public String breed;
  public double size;


  public void bark( ) {
    if (size > 1.0) {
      System.out.println( "Ruff" );
    } else {
      System.out.println( "bark" );
    }
  }
}
```

Java

```
public class Main {
  public static void main( String[] args ) {
    Dog john = new Dog( );
    john.breed = "Bulldog";
    john.size = 2.0;
    Dog lucky = new Dog( );
    lucky.breed = "Terrier";
    lucky.size = 0.5;

    john.bark( ); // Ruff
    lucky.bark( ); // bark
  }
}
```

How Does it Work in Java?

Make John Bark

Java

```
public class Dog {
  public String breed;
  public double size;


  public void bark( ) {
    if (size > 1.0) {
      System.out.println( "Ruff" );
    } else {
      System.out.println( "bark" );
    }
  }
}
```

Java

```
public class Main {
  public static void main( String[] args ) {
    Dog john = new Dog( );
    john.breed = "Bulldog";
    john.size = 2.0;
    Dog lucky = new Dog( );
    lucky.breed = "Terrier";
    lucky.size = 0.5;

    john.bark( ); // Ruff
    lucky.bark( ); // bark
  }
}
```

How Does it Work in Java?

Make Lucky Bark

Java

```
public class Dog {
  public String breed;
  public double size;


  public void bark( ) {
    if (size > 1.0) {
      System.out.println( "Ruff" );
    } else {
      System.out.println( "bark" );
    }
  }
}
```

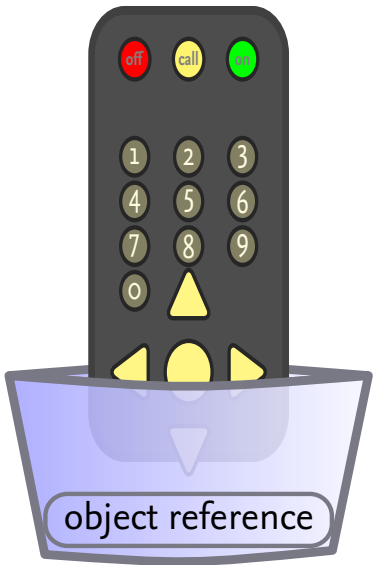
Java

```
public class Main {
  public static void main( String[] args ) {
    Dog john = new Dog( );
    john.breed = "Bulldog";
    john.size = 2.0;
    Dog lucky = new Dog( );
    lucky.breed = "Terrier";
    lucky.size = 0.5;

    john.bark( ); // Ruff
    lucky.bark( ); // bark
  }
}
```

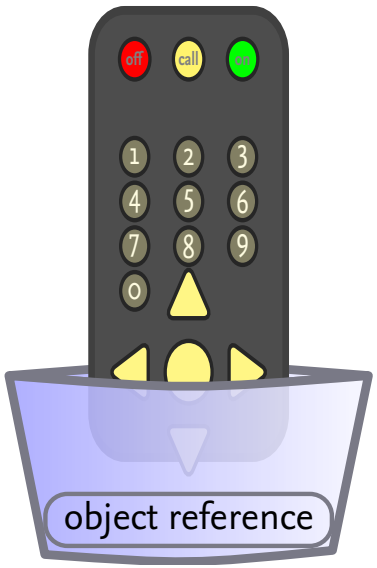
Interview with an Object Reference Variable

So, what's it like to be an object reference?



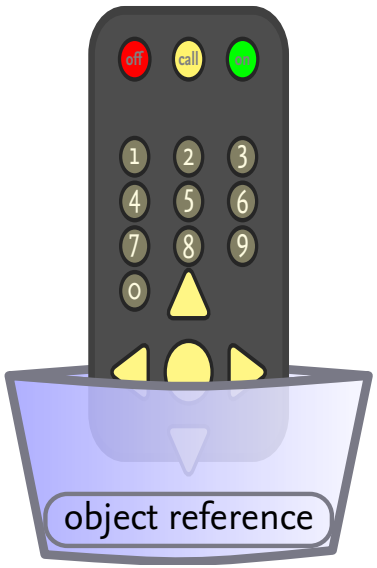
Interview with an Object Reference Variable

I'm a remote control.
You can program me to control different objects.



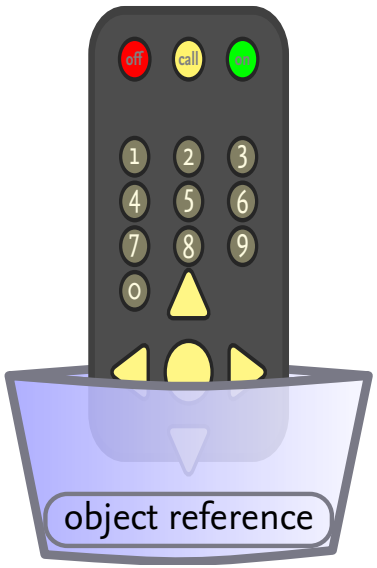
Interview with an Object Reference Variable

Cool. Do you mean different kinds of objects?
For example, a `Cat` object and a `House` object?



Interview with an Object Reference Variable

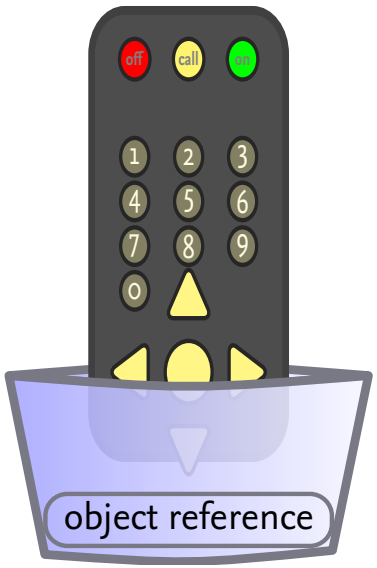
```
remote = new Cat( );
remote = new House( );?
```



Interview with an Object Reference Variable

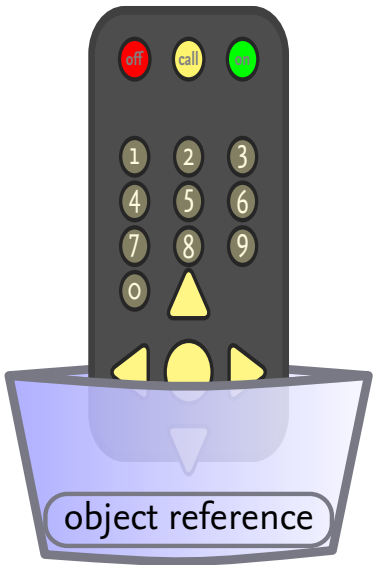
Interview with an Object Reference Variable

Of course not. I'm well behaved.
I've a type, like!



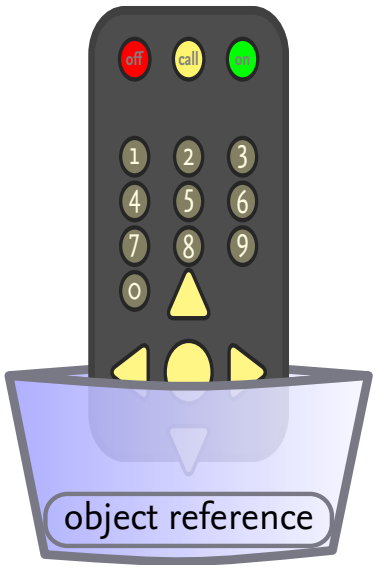
Interview with an Object Reference Variable

```
Cat remote;
remote = new House( ); // not allowed
```



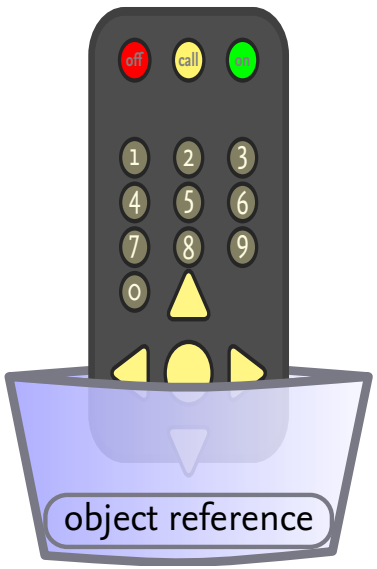
Interview with an Object Reference Variable

But I don't understand.
They told me you can control
several, different, objects.

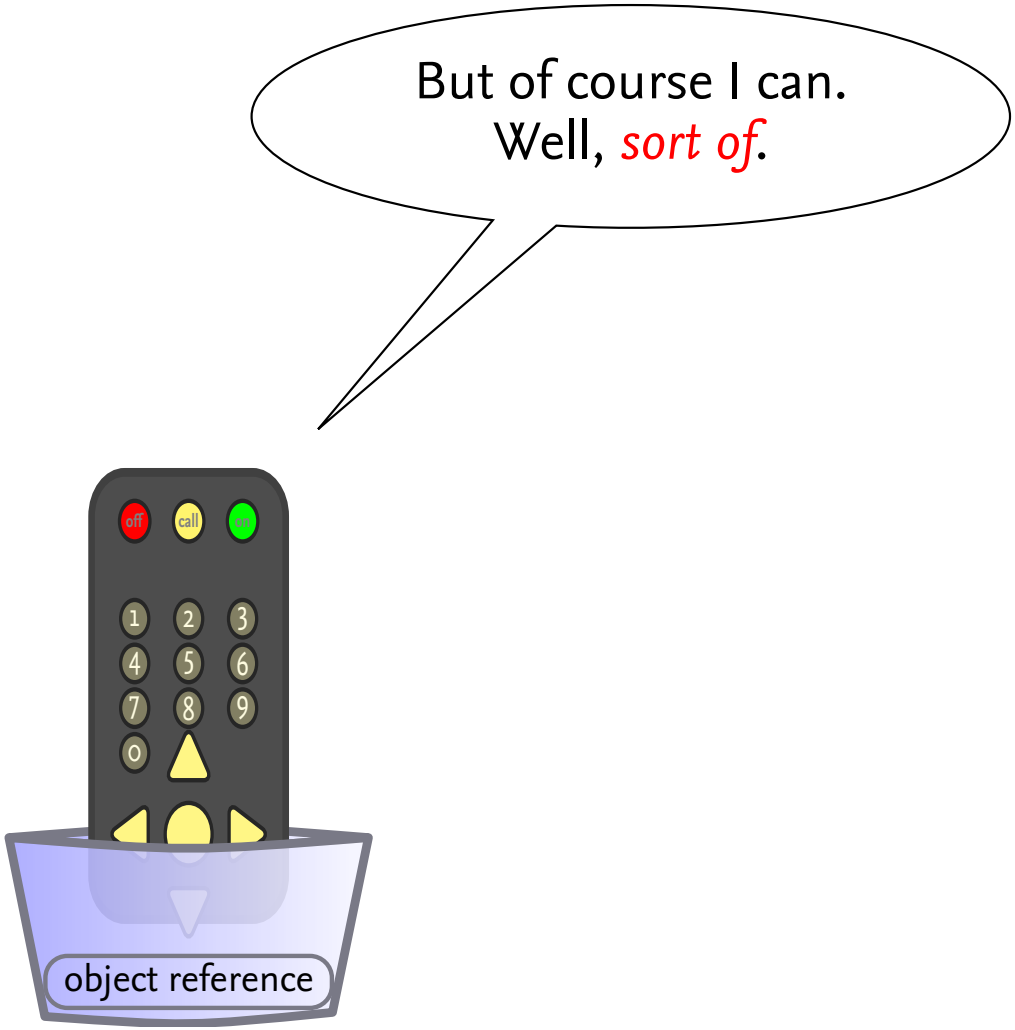


Interview with an Object Reference Variable

The people they send out
for interviews these days....

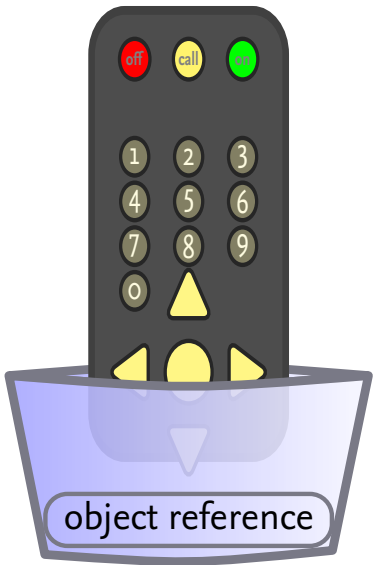


Interview with an Object Reference Variable



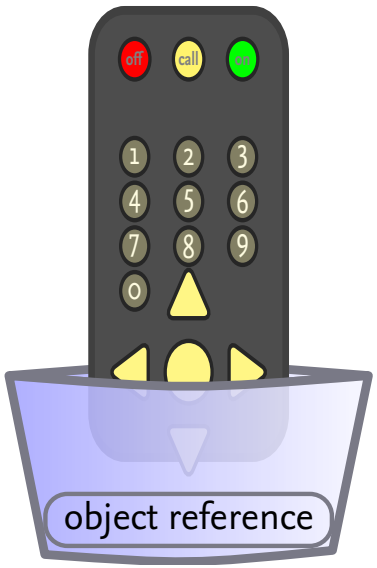
Interview with an Object Reference Variable

```
Cat remote = new Cat( );
Cat other = new Cat( );
remote = other;
```



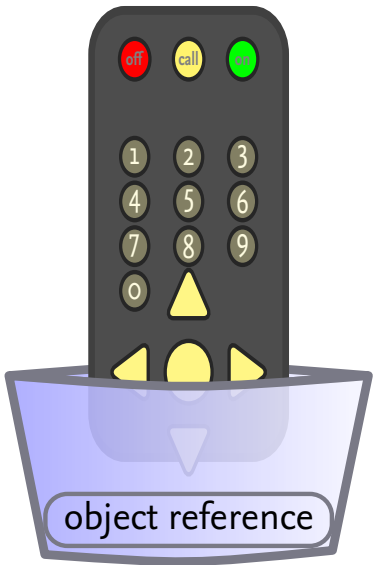
Interview with an Object Reference Variable

Ah, I see, but what about that “sort of?”



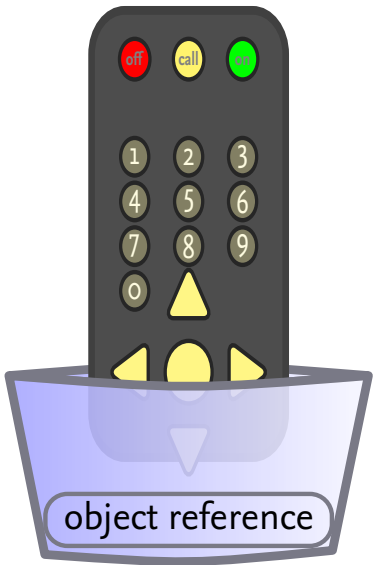
Interview with an Object Reference Variable

There are two things really. The first is when I'm `final`: `final` variables can only take one value.



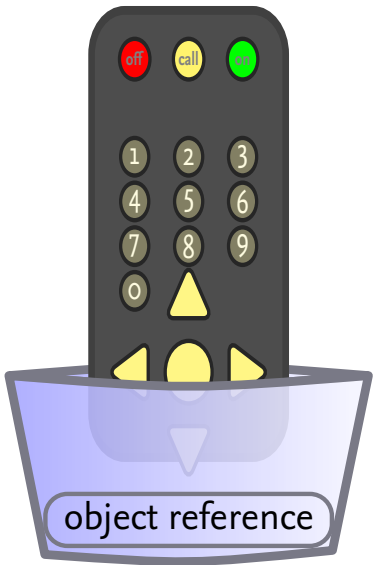
Interview with an Object Reference Variable

```
final Dog remote = new Dog( );
remote = new Dog( ); // not allowed
```



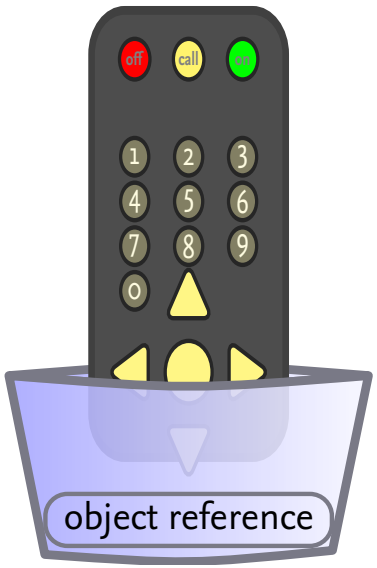
Interview with an Object Reference Variable

The second thing is when my type is `Object`.
As `Object` I can take *any object* reference value.



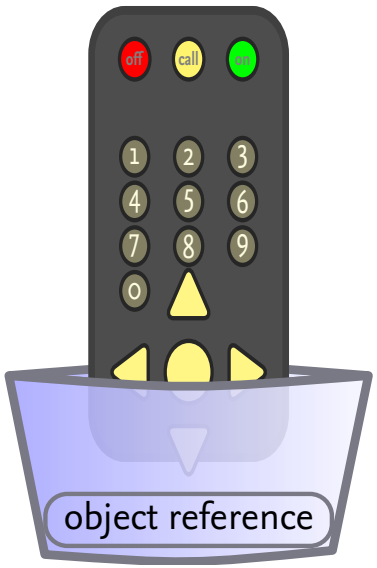
Interview with an Object Reference Variable

```
Object remote = new Cat( );
remote = new Dog( );
```



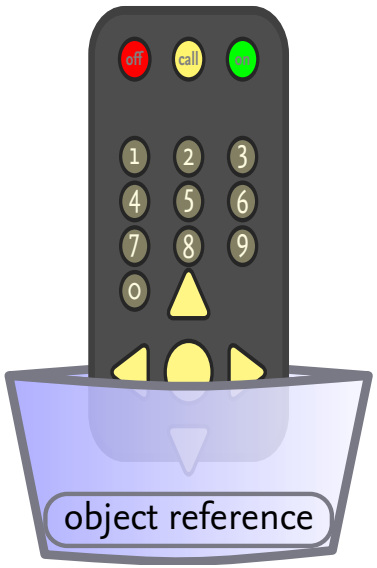
Interview with an Object Reference Variable

The `Object` type makes me paranoid. It's just not safe. You can be assigned any object reference value. Including reference values that aren't safe.



Interview with an Object Reference Variable

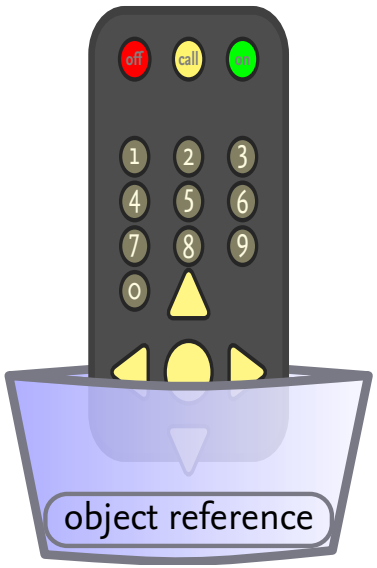
I get it. You're saying that types help prevent certain errors.



Interview with an Object Reference Variable

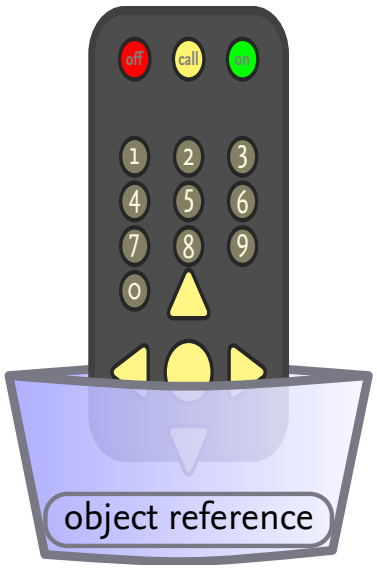
Interview with an Object Reference Variable

This is the right moment.
I've got to ask him now.

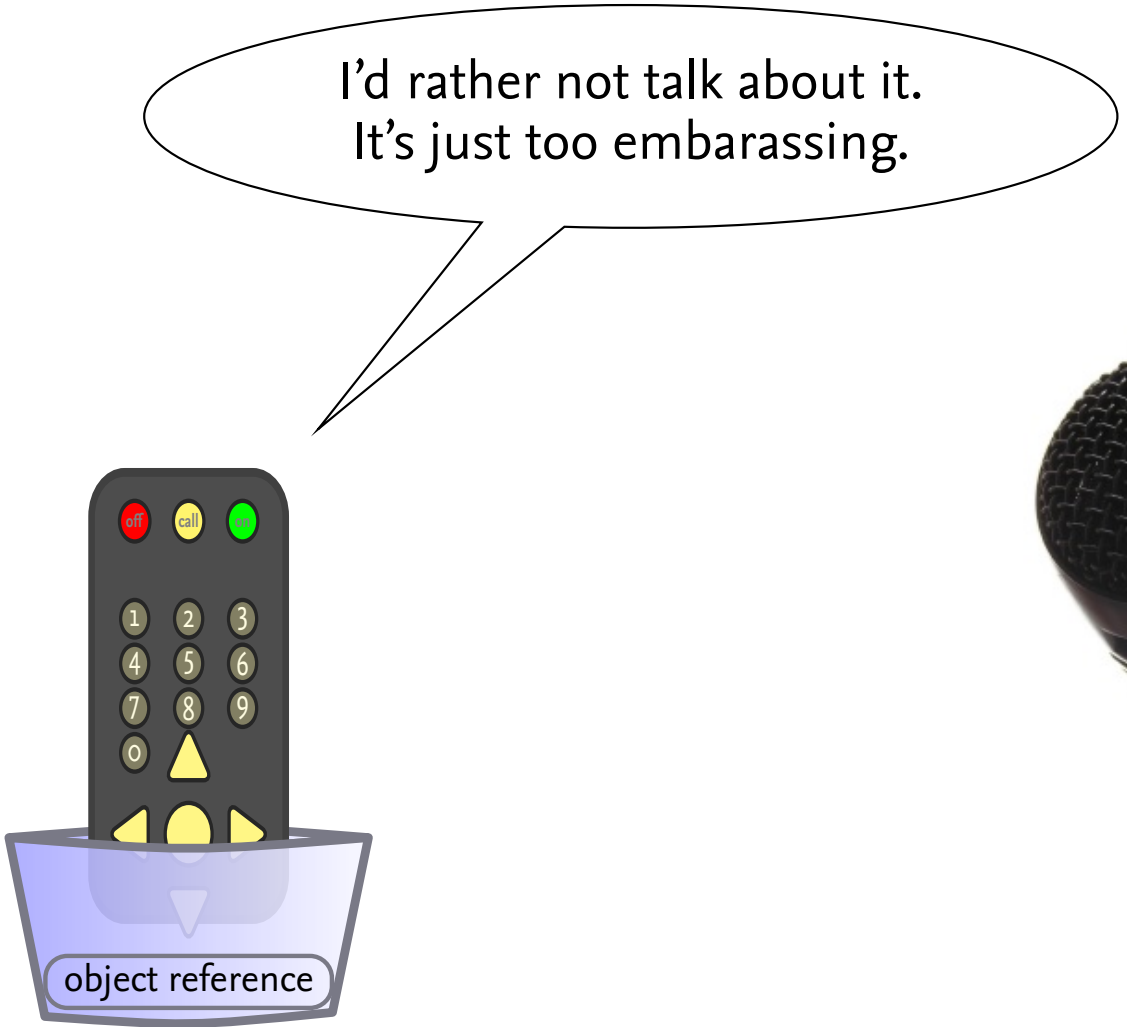


Interview with an Object Reference Variable

Can you say something about those `null` values? I've heard they're a bit of a taboo subject among object reference variables.

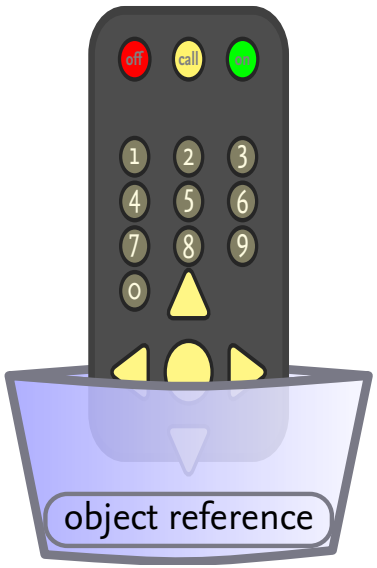


Interview with an Object Reference Variable



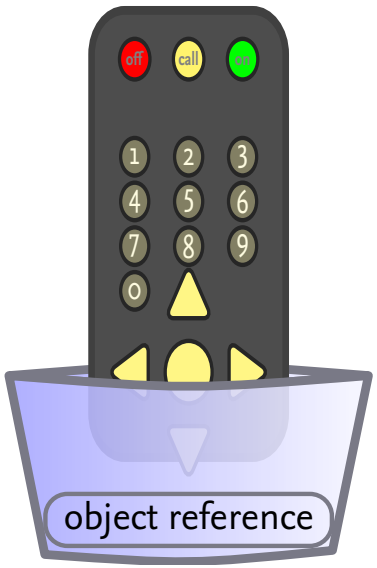
Interview with an Object Reference Variable

I'm sorry to hear that, but wouldn't you at least give it a try? Our audience would be really interested in this.



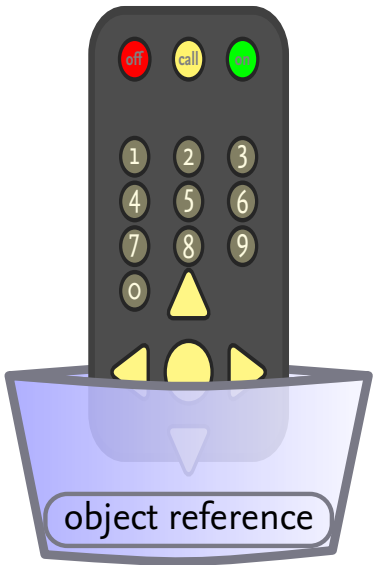
Interview with an Object Reference Variable

Well, *any* object reference variable
can be assigned the value `null`.
It's the worst thing that can happen to us.



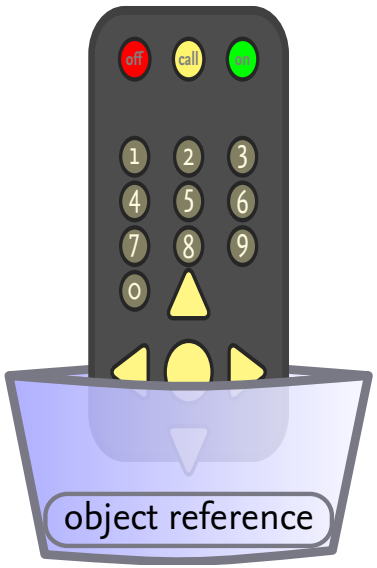
Interview with an Object Reference Variable

```
Dog remote = null;
Cat cat = null;
```

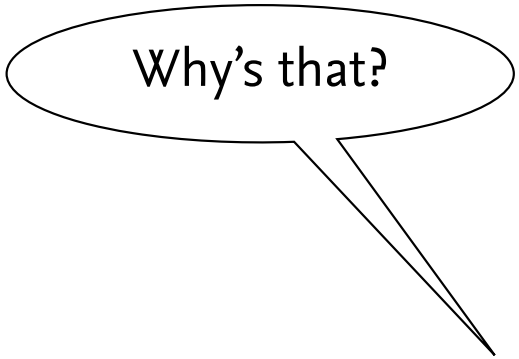


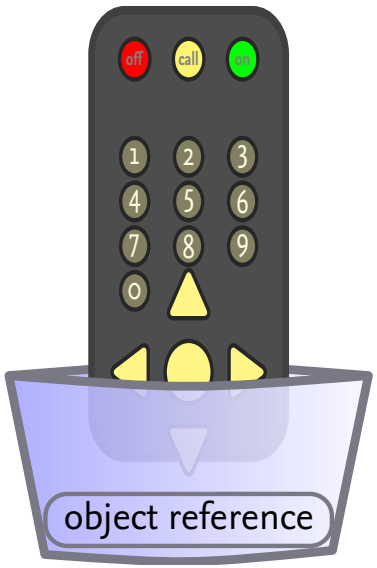
Interview with an Object Reference Variable

Sensational!



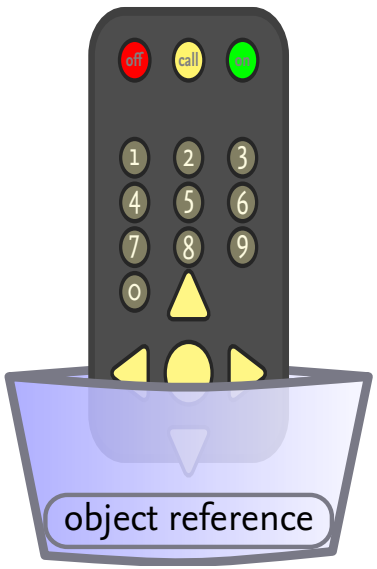
Interview with an Object Reference Variable





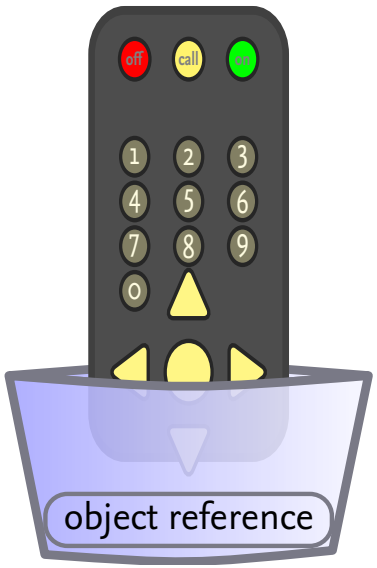
Interview with an Object Reference Variable

Well, when that happens, you're nothing. It's like unsubscribing from SKY: When you do that, your SKY remote becomes useless.



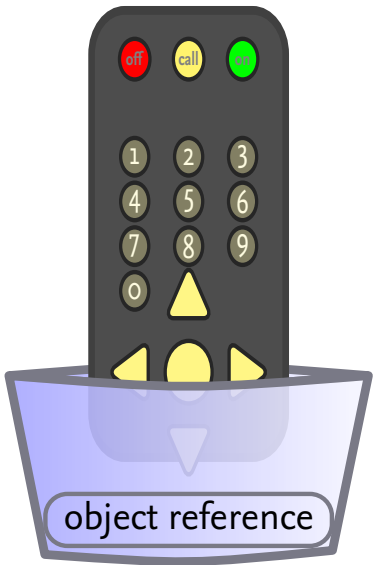
Interview with an Object Reference Variable

For us object reference variables it's even worse. When we are `null` we're not even allowed to *try* control objects.



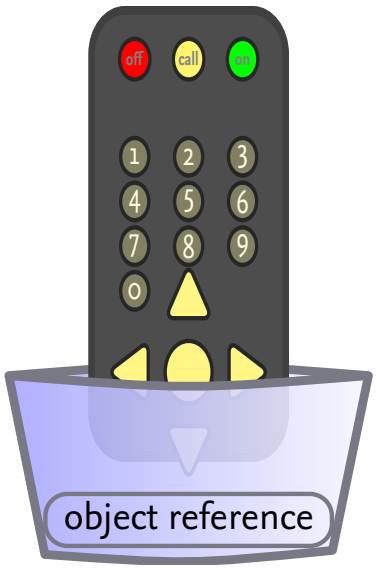
Interview with an Object Reference Variable

```
Dog remote = null;
remote.bark( ); // run-time error.
```



Interview with an Object Reference Variable

Well thanks. That was very insightful.
Have a nice day.



Interview with an Object Reference Variable

For Friday: Study Chapter 1

Acknowledgements

- This lecture is based on ideas from Sierra, and Bates [2004, Chapter 2].

About this Document

- This document was created with `pdflatex`.
- The LaTeX document class is `beamer`.