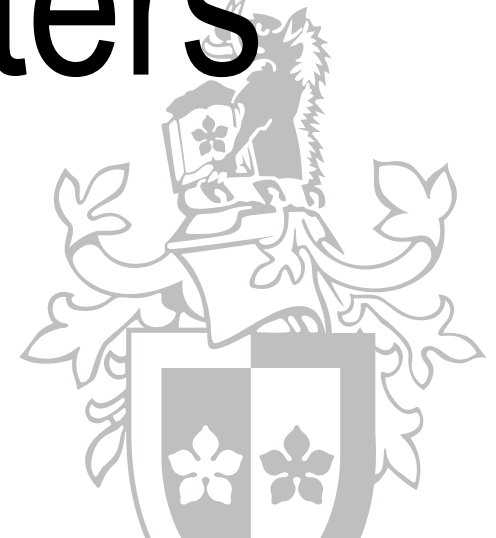


Delegation: Interfaces and Method Pointers

Charlotte Pierce



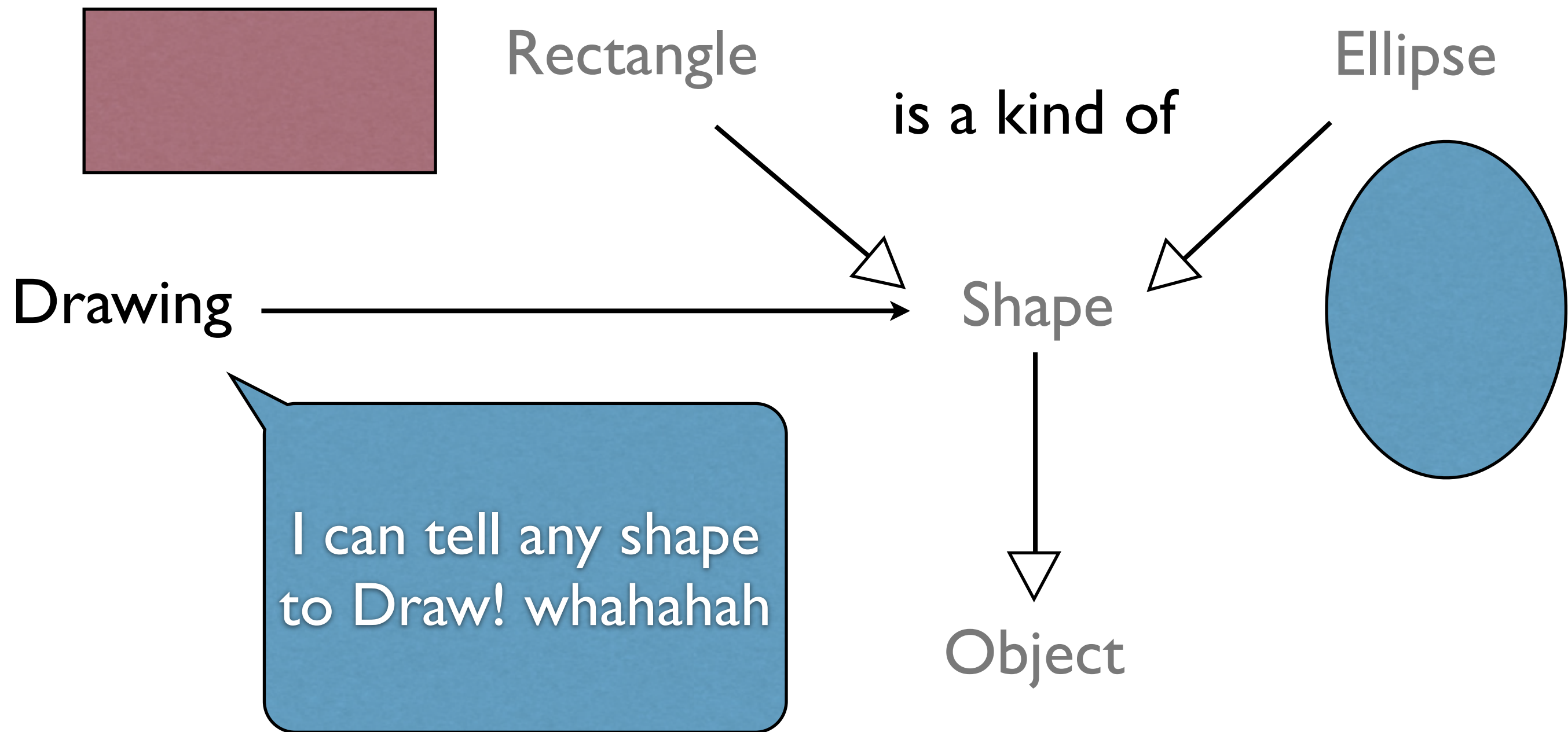
SWIN
BUR
NE

SWINBURNE
UNIVERSITY OF
TECHNOLOGY

Object oriented programs contain objects that know and can do things



Developers use inheritance to create families of types with common features



What about cases where an object wants to interact, but not with a family of related types

Sorter

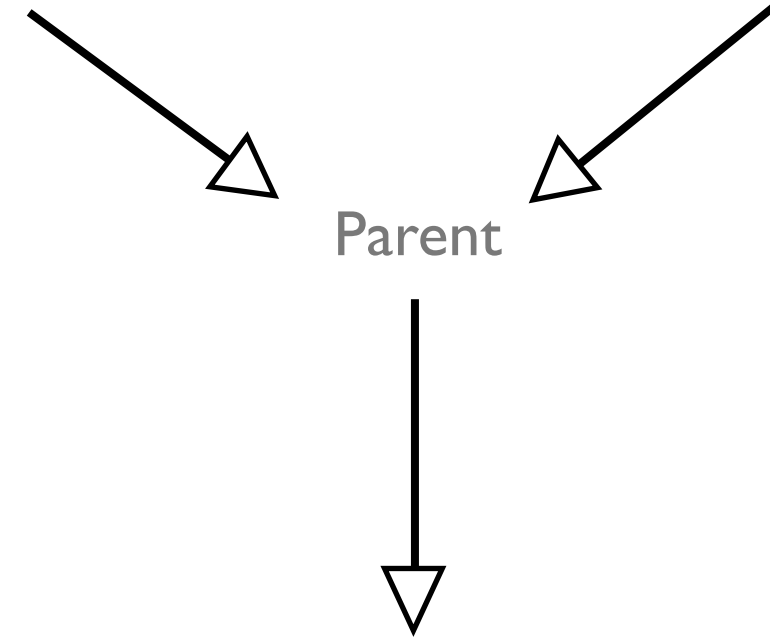
I want to sort ... as
many kinds as
possible...

Child 1

Child 2

Parent

Object



Ideally the object should be able to say what features they need...

Sorter



I can sort anything
with these features
(anything
Comparable)

Something Comparable must
be able to...

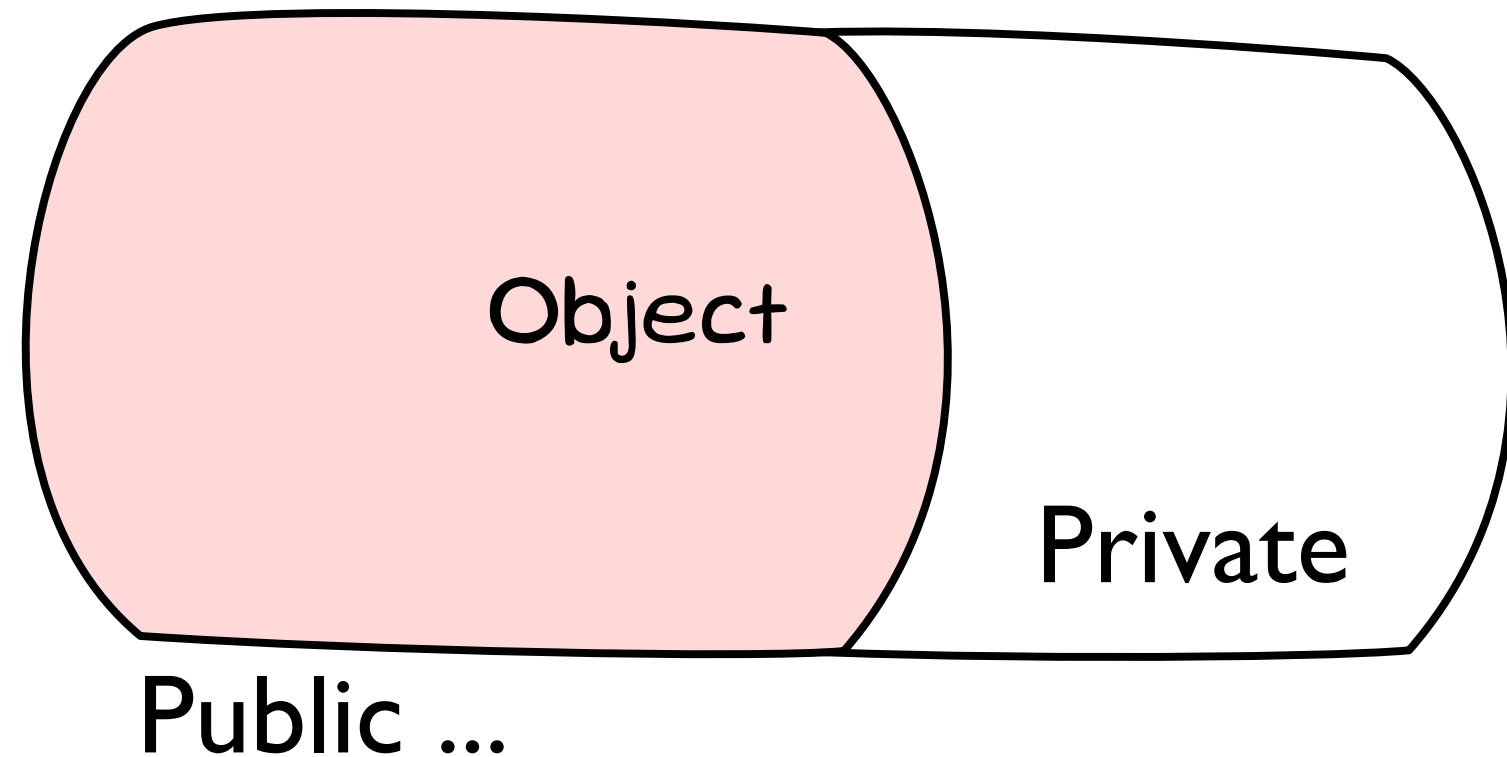
Compare itself **To** another object

Use interfaces or method pointers
to define the features you need

Use an interface to define the
features you need

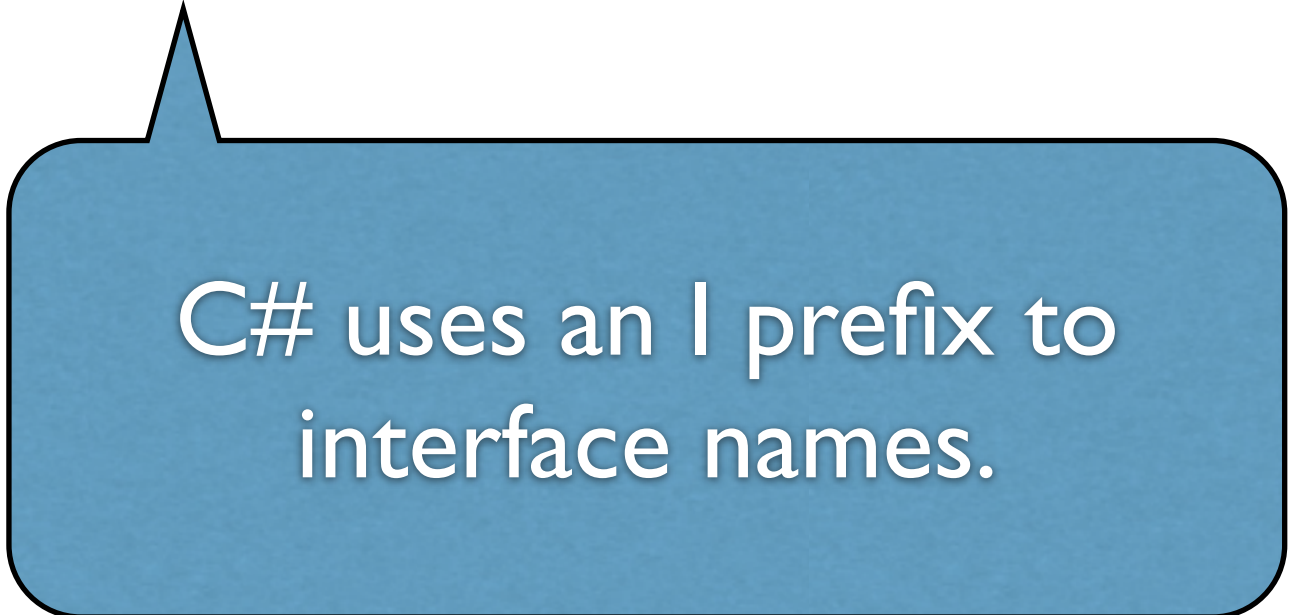
Specify the features that implementing classes must provide

To be Comparable you must have an "int Compare(...)" method...



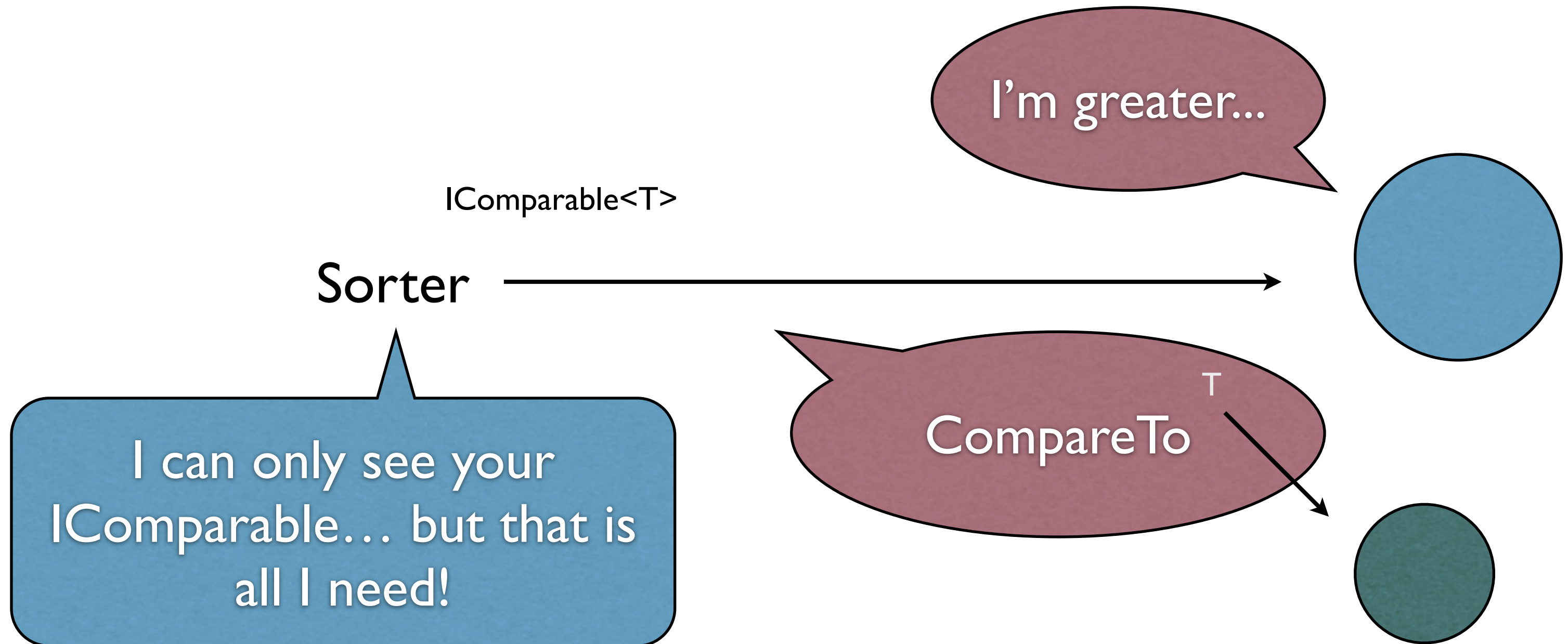
List these required features in the Interface declaration

```
public interface IComparable<in T>
{
    int Compare(T other);
}
```



C# uses an I prefix to interface names.

Use the interface and access these features on whatever is supplied to you!

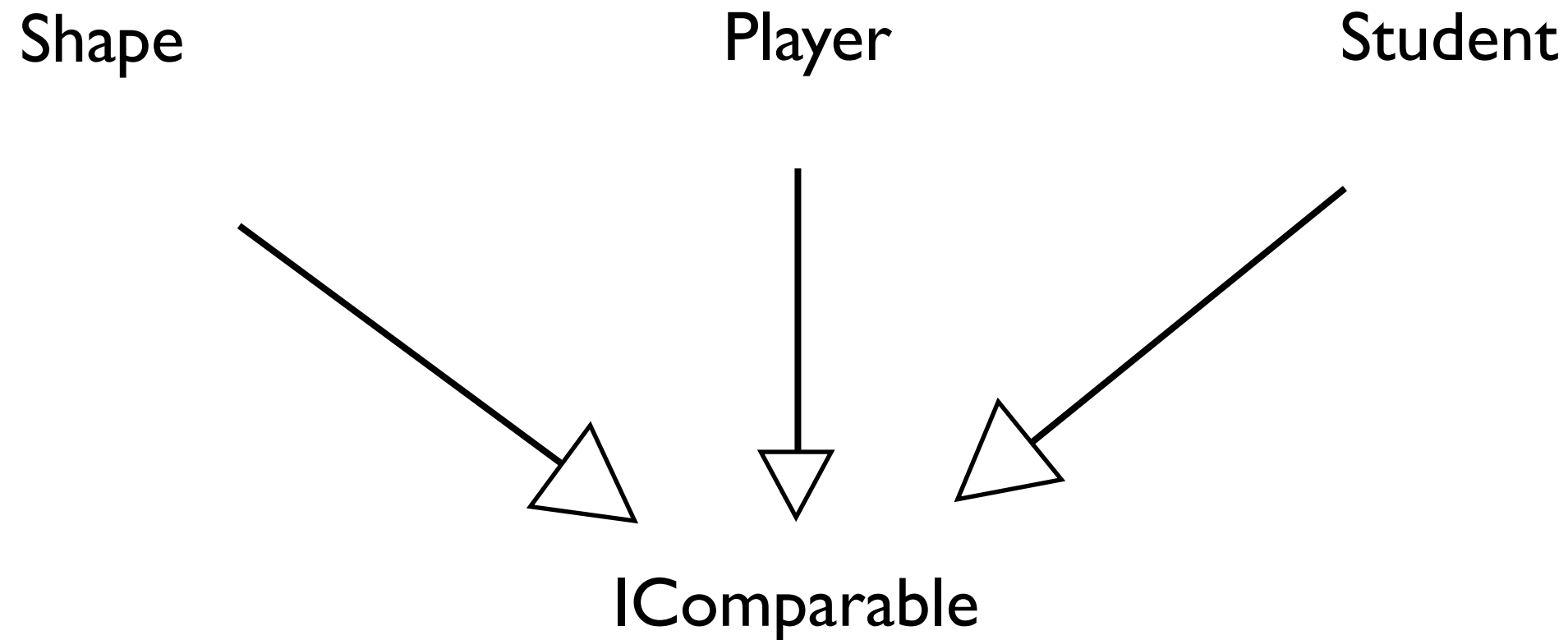


Implement the interface if you want
the services provided

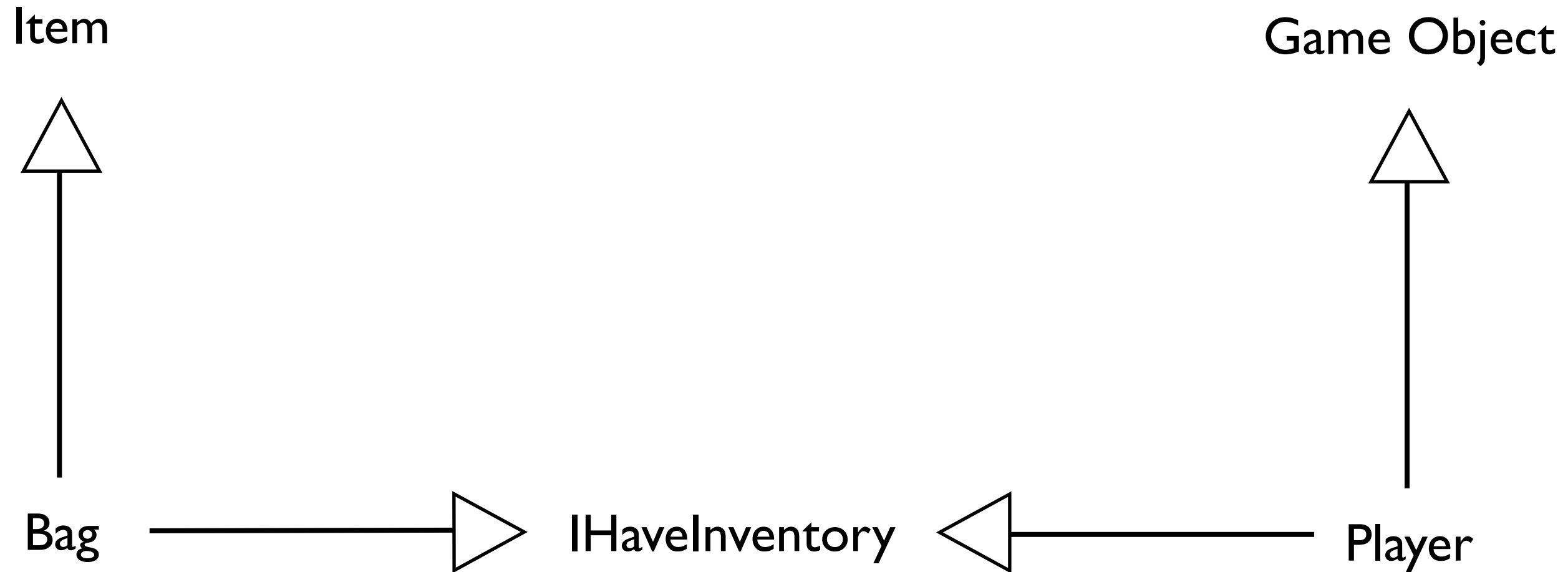
Implement the interface for any role that wants to be used by the other object

```
public class Student : IComparable<Student>
{
    public int Compare(Student other) {...}
}
```

Polymorphism means objects of this type can now be used anywhere the interface is needed



Classes can inherit from one class, but can implement many interfaces



Demo - Using an Interface

Consider delegates (method pointers)
where you want just a single method

In the delegate, define the kind of method you want

```
public delegate void EventHandler(object sender);
```



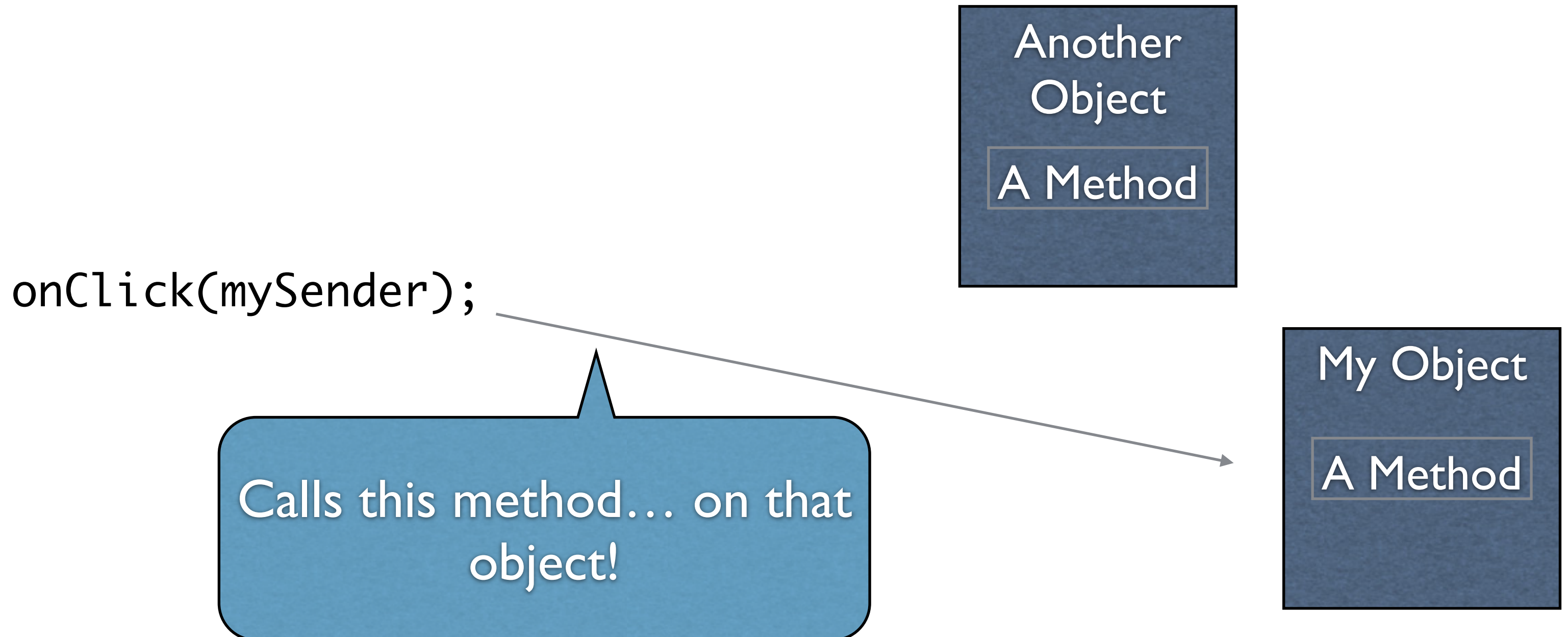
Any method that looks like
this... exclude name

Use the delegate type to create variables that point to methods

```
EventHandler onClick = myObject.AMethod; // no ( )
```



Call delegate variable to execute the methods they refer to



Will interfaces and delegates help
you make reusable utilities?

Standard inheritance needs you to
have a family of related types

Use interfaces or method pointers
to define the features you need

Interfaces and delegates allow you
to access features in a flexible way

Interfaces and Delegates