Reviewing Object Oriented Programming Principles

Charlotte Pierce

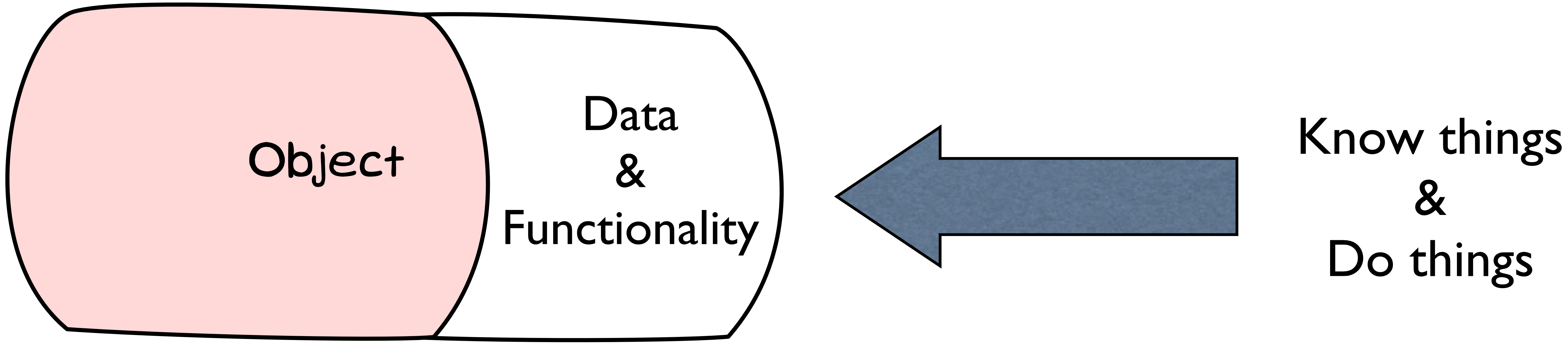
SWINBURNE UNIVERSITY OF TECHNOLOGY

Semester Test

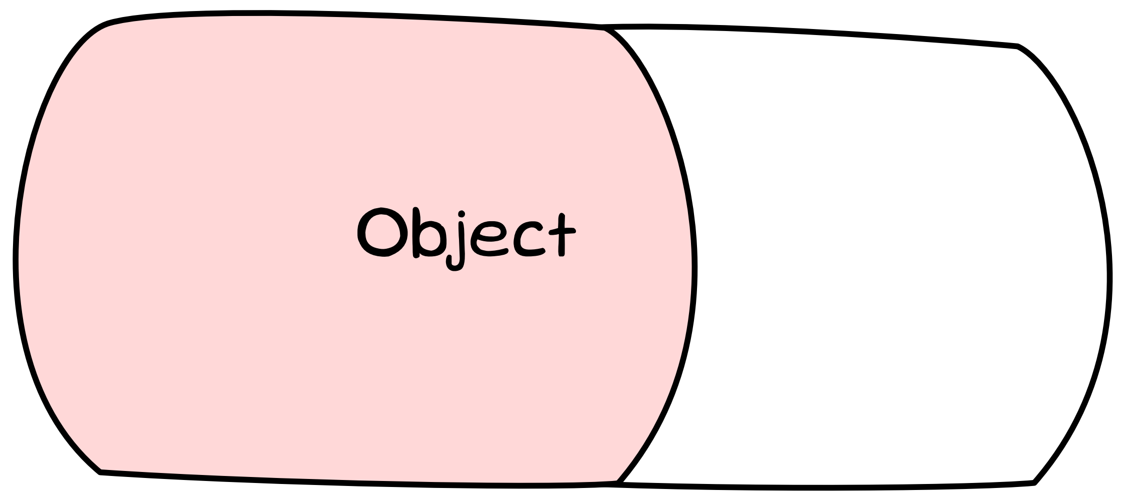
- Expectations:
  - you know the core OO principles, and can apply them in C#
  - you can interpret and possibly modify UML
  - you know C# syntax (you will be writing code)
  - you know the commonly used methods and properties of collection classes like List and Dictionary

You can’t fail the test!

Object oriented programming involves creating objects that know and do things



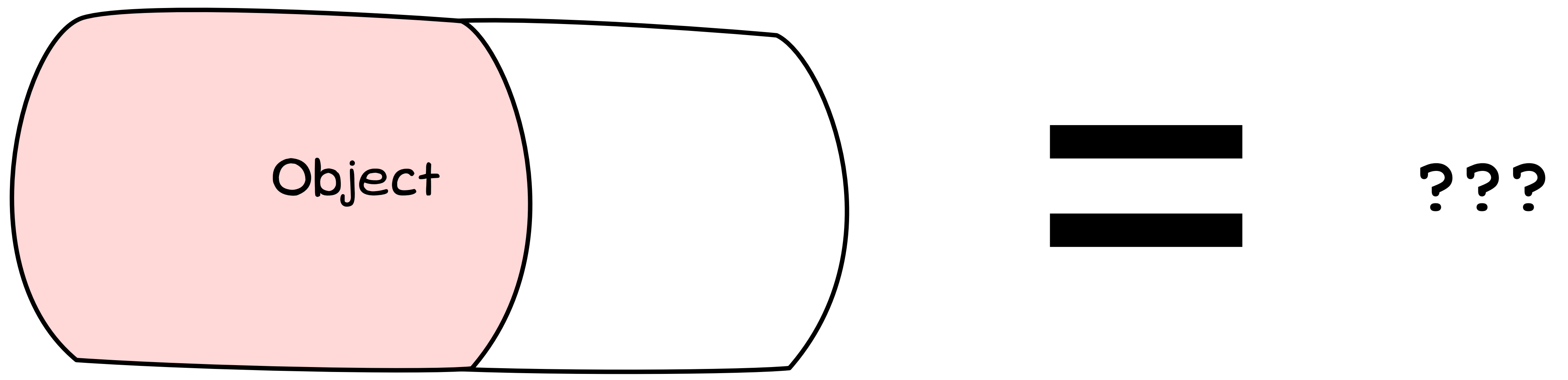
To succeed at OOP, you need to understand objects and how they work



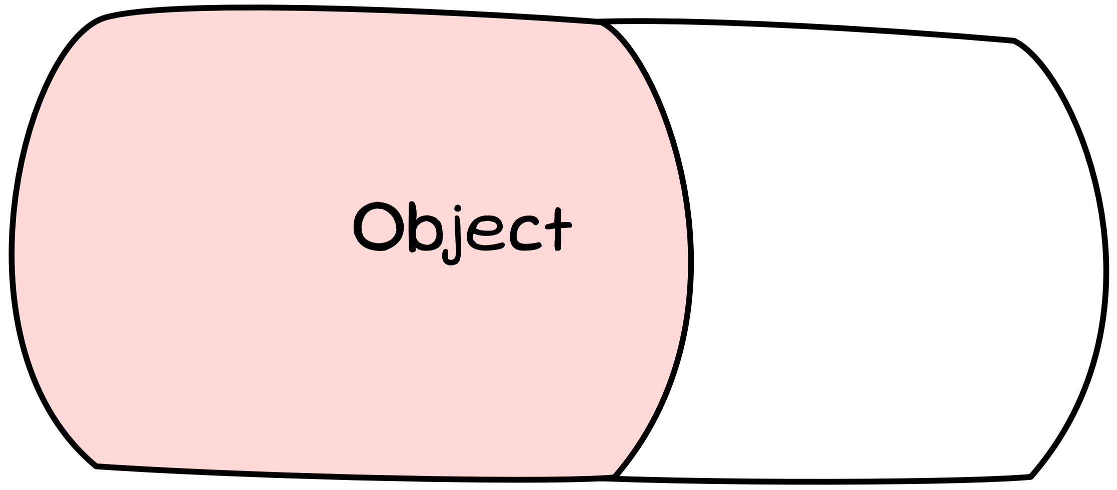
>

```
public class Location
{
  ...
}
```

Without clear understanding, its hard to see how objects work and hard to explain



A clear understanding makes explaining these principles and designing programs easier



=

```
public class Location
{
  … MovePlayer …
  … LongDescription …
  … Locate …
  … Inventory …
}
```

See how profound “objects know and do things” is in relation to the OO principles

Abstraction

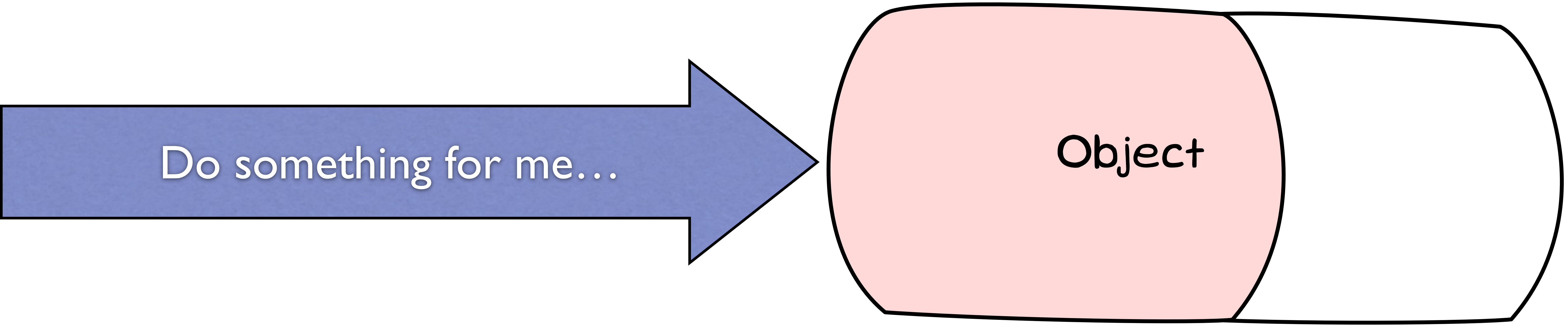
Encapsulation

Inheritance

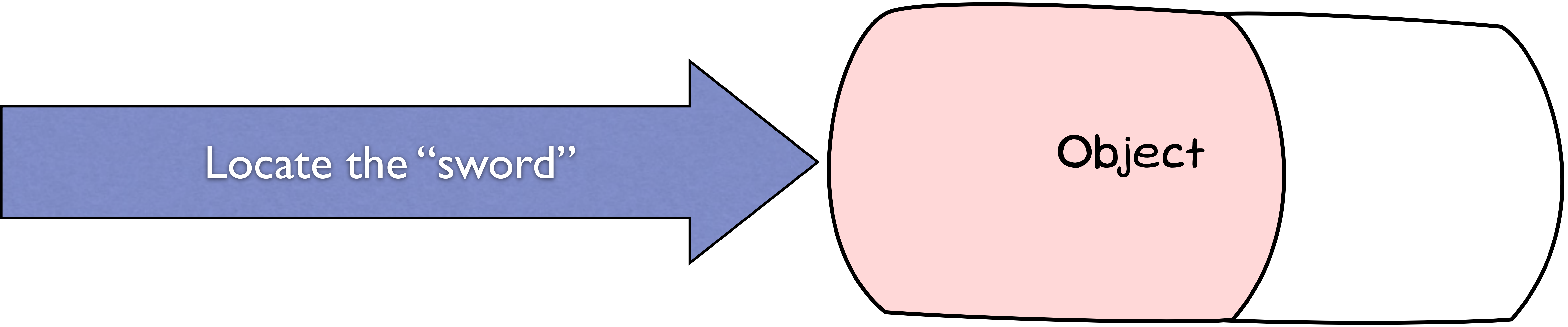
Polymorphism

Objects start with **encapsulation**:
things that contain
knowledge and functionality

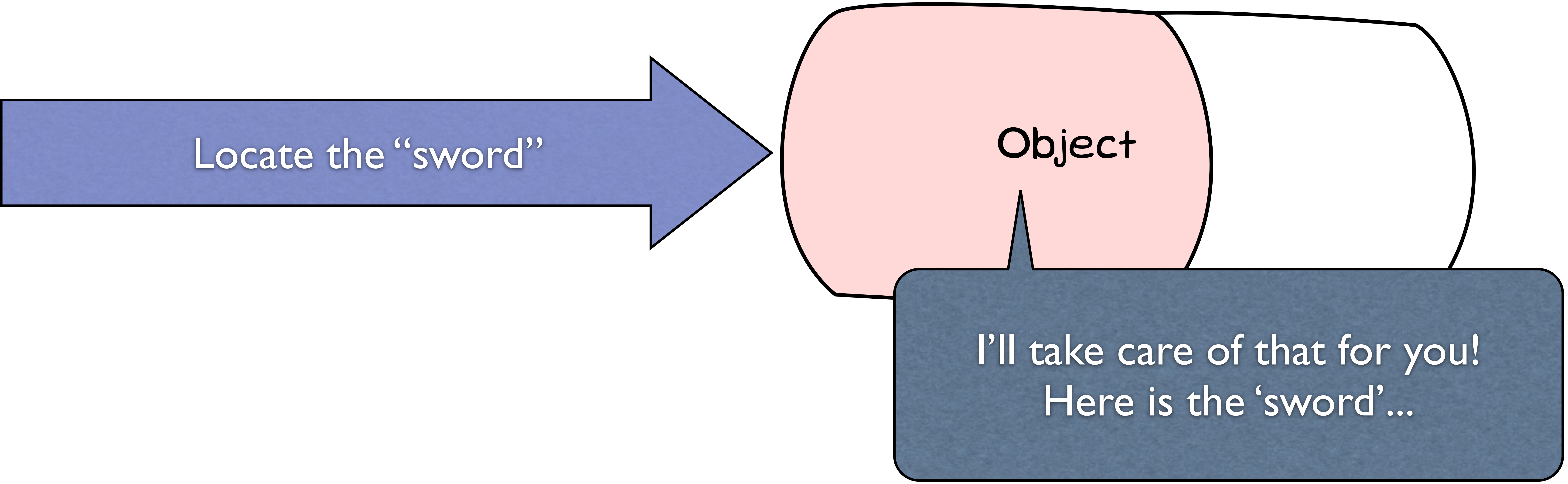
Objects exist as entities that you can interact with



Externally, you don't need to worry about how objects work inside



Tell the object what to do, and let it take responsibility for getting it done!

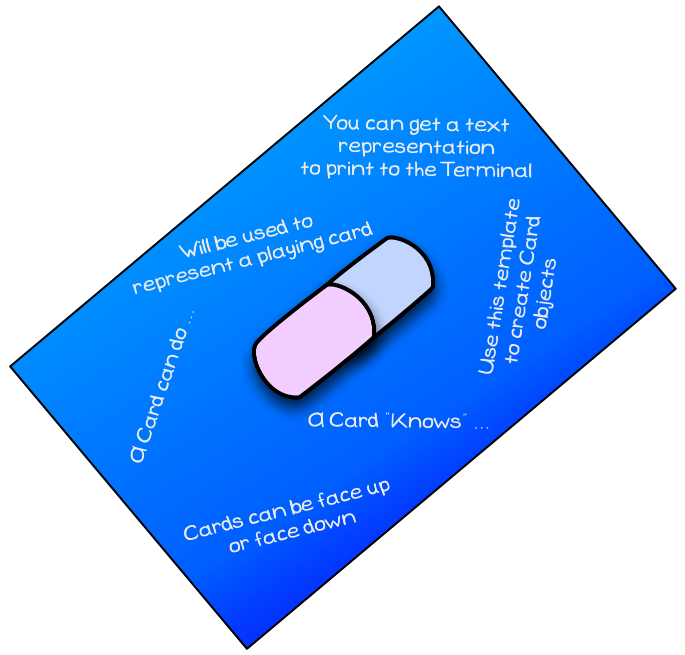


Abstraction helps identify classifications, roles, responsibilities, and collaborations

Build the *things* for your program by abstracting them from the "*real world*"

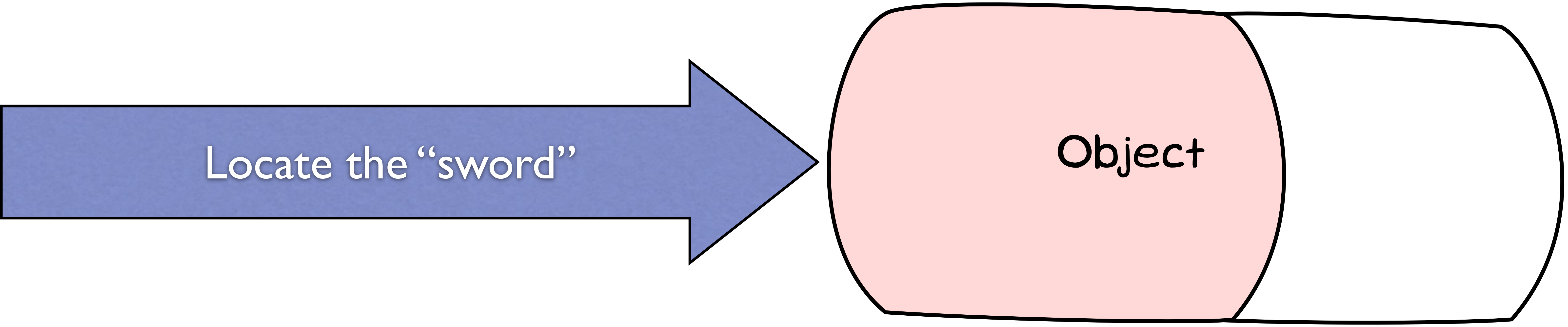
}

Specification for a Card

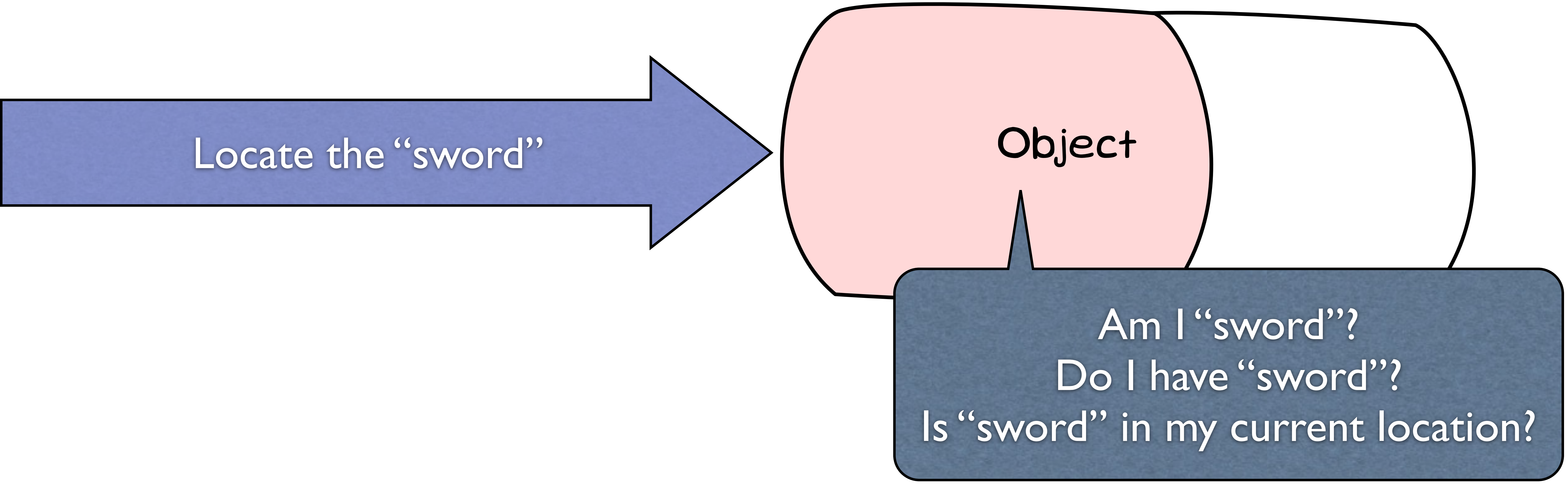

You can get a text representation to print to the Terminal
Will be used to represent a playing card
Use this template to create Card objects
a Card can do ...
a Card "Knows" ...
Cards can be face up or face down

Classification

Determine responsibilities for classes: what messages do these objects respond to?

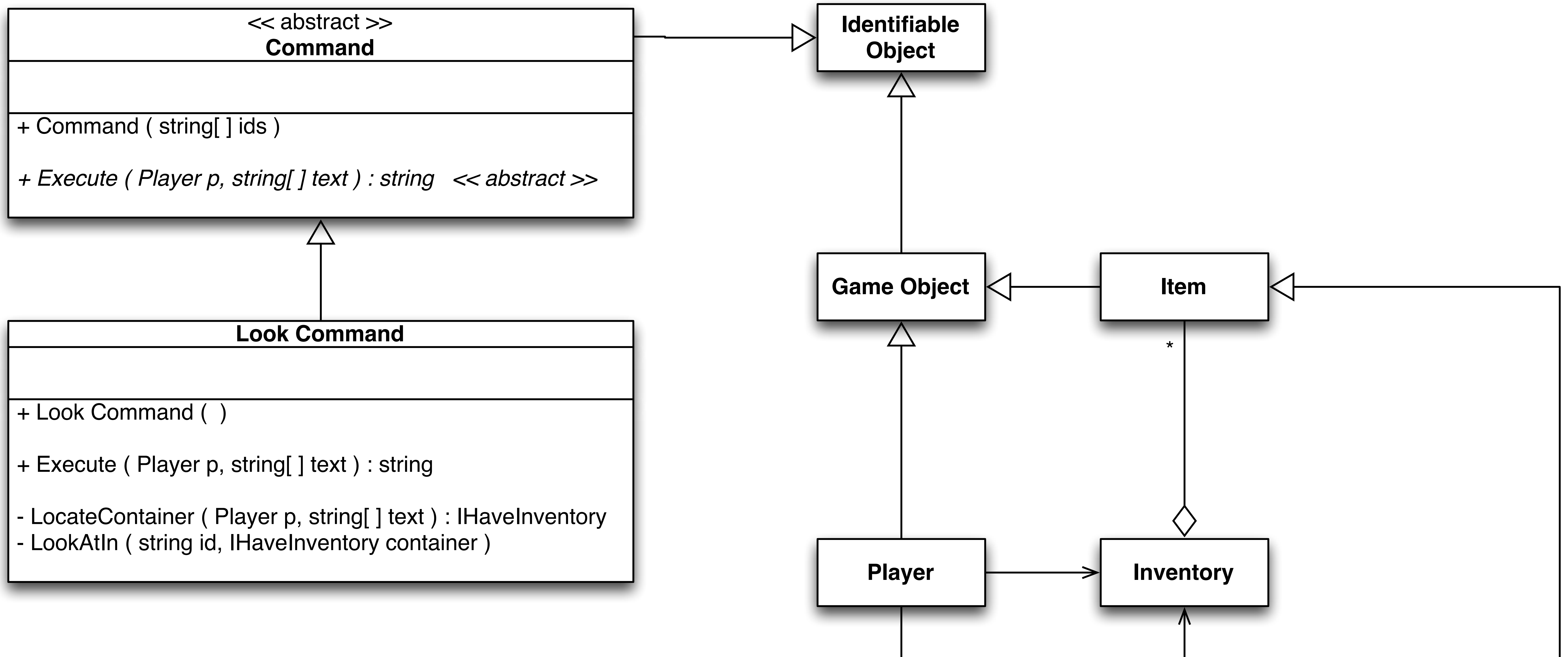


Identify the need for any collaborations with other objects



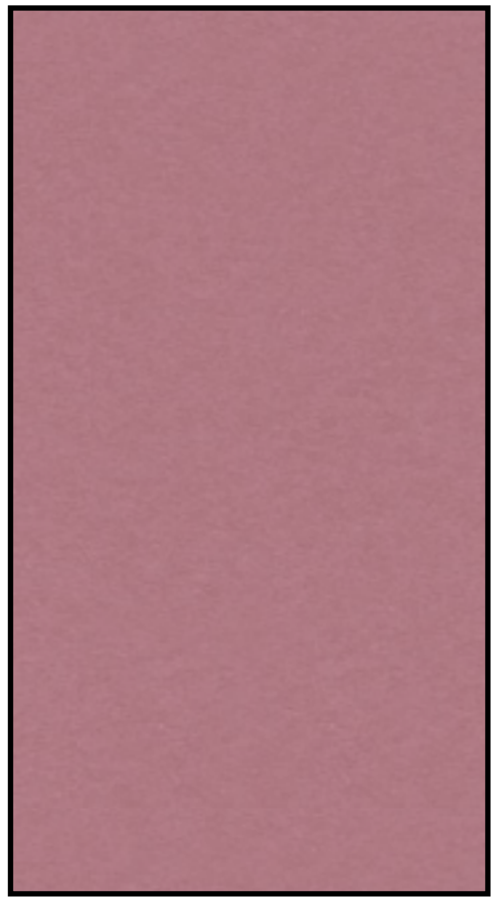
Use **inheritance** to create generalised and specialised families of classes

Create families of related classes, reusing functionality from parent classes



<< abstract >>
Command
+ Command (string[ ] ids)
+ Execute (Player p, string[ ] text) : string << abstract >>
Look Command
+ Look Command ( )
+ Execute (Player p, string[ ] text) : string
- LocateContainer (Player p, string[ ] text) : IHaveInventory
- LookAtIn (string id, IHaveInventory container)
Identifiable Object
Game Object
Item
*
Player
Inventory

Objects encapsulate a combination of features: some inherited some specific

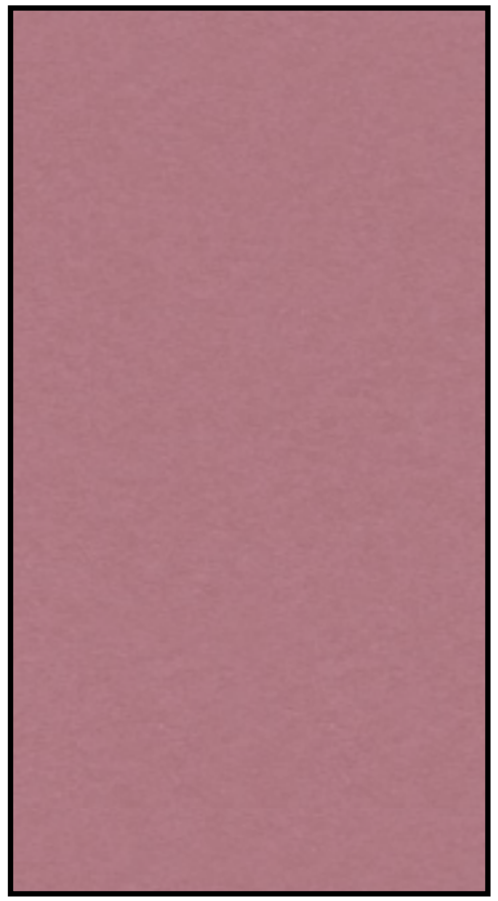


Inherits Object characteristics

Inherits Shape characteristics

Includes Rectangle characteristics

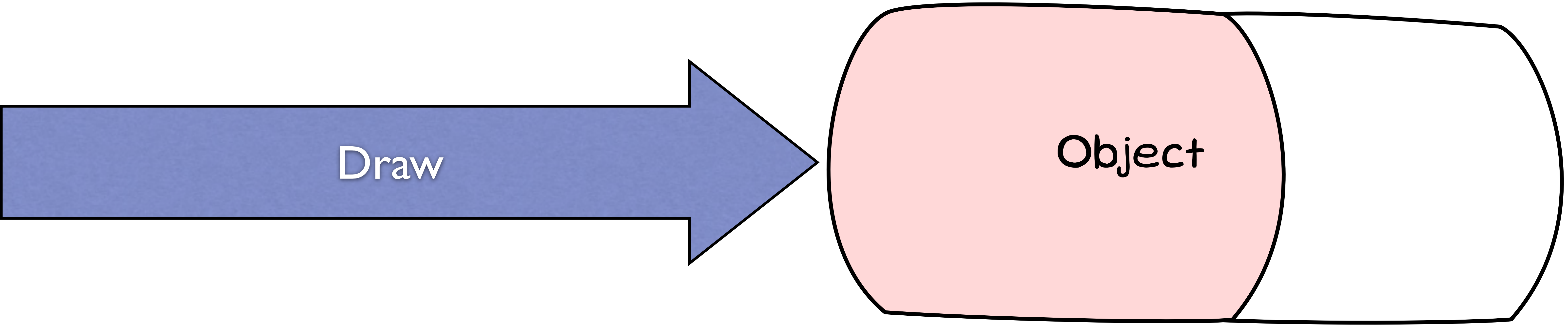
Customise inherited features where differences occur



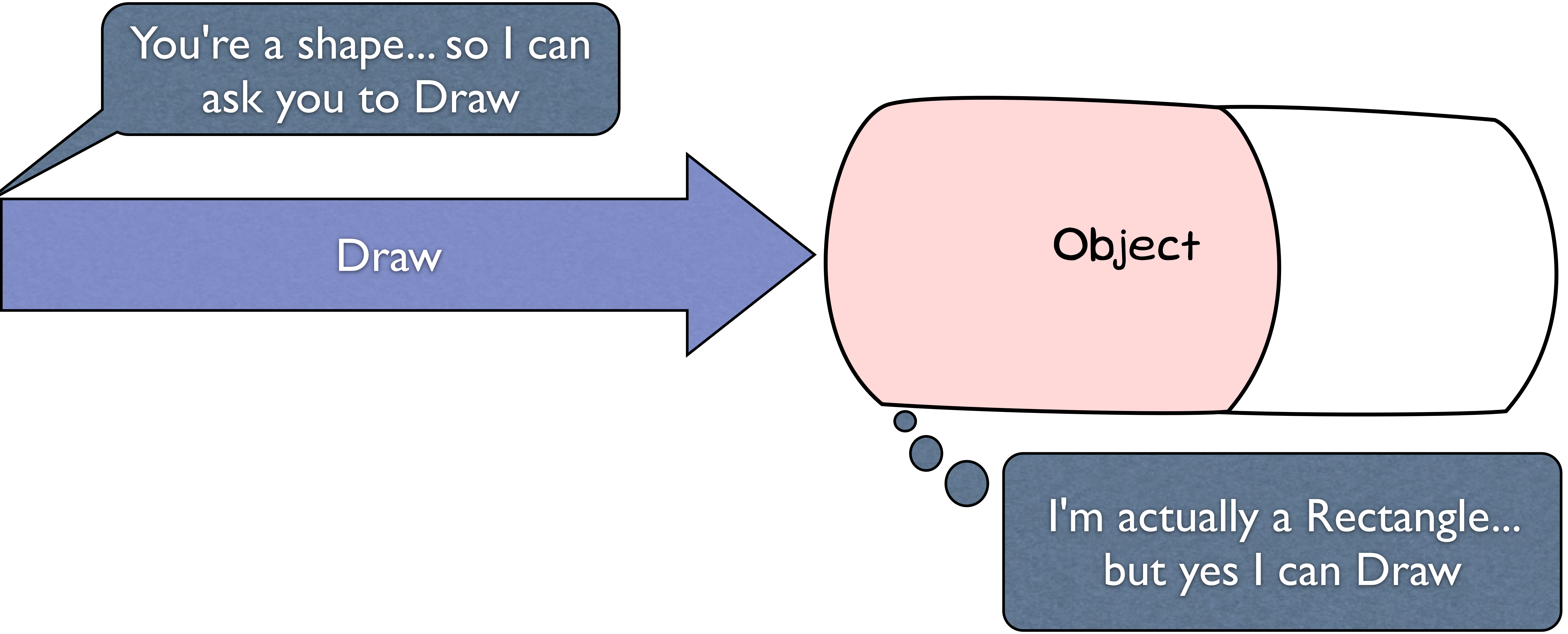
Draws like a Rectangle

Tie it all together, and add flexibility where needed with **polymorphism**

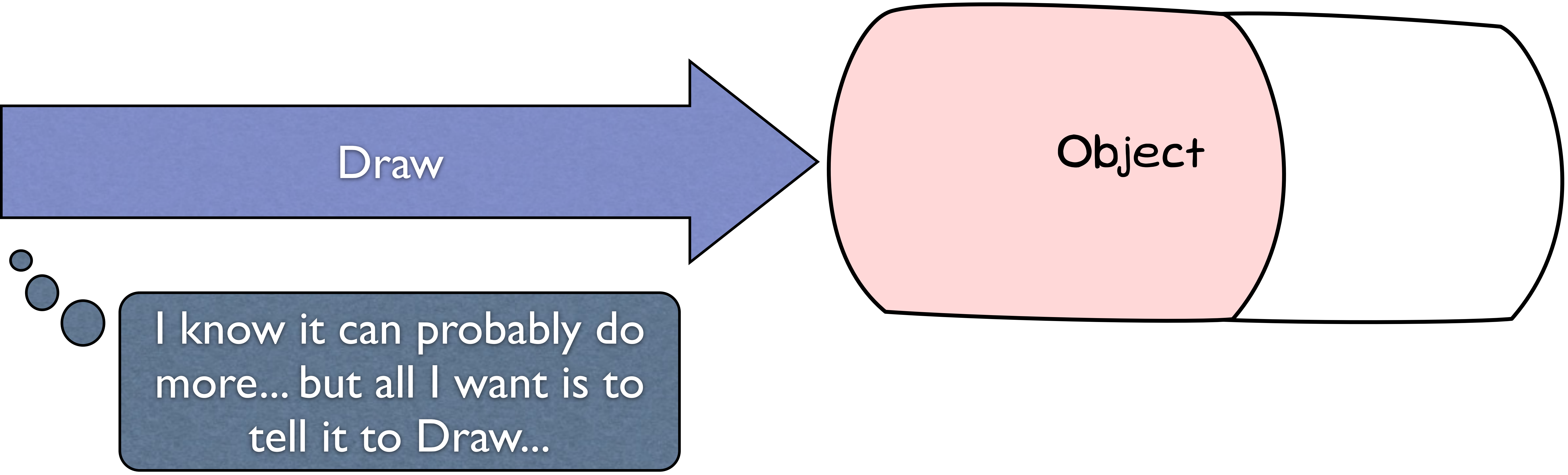
Remember objects encapsulate a range of features: some of which are inherited



A single variable reference can access multiple unique implementations


You're a shape... so I can ask you to Draw
Draw
Object
I'm actually a Rectangle... but yes I can Draw

When selecting a variable type, choose the most general type that will still be suitable


Draw
Object
I know it can probably do more... but all I want is to tell it to Draw...

Will understanding these principles help you create better object oriented programs?

Four principles underly everything in object oriented programming

See how profound “objects know and do things” is in relation to the OO principles

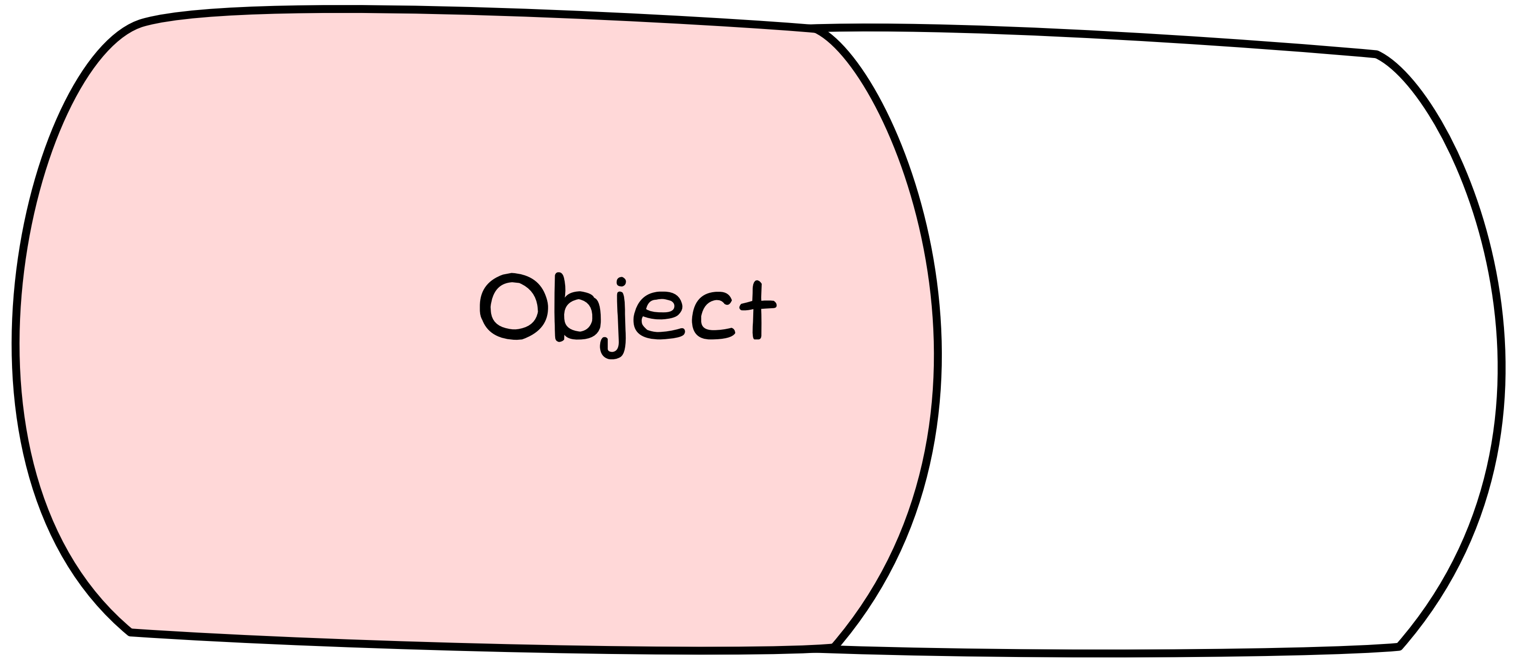
Abstraction

Encapsulation

Inheritance

Polymorphism

Design any program using an understanding of these ideas together with basic control flow logic



Encapsulation, abstraction, inheritance, and polymorphism make OOP possible