Thinking in Objects

- What is an Object?
- What is a Class?
- The Three Pillars of OO

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Benefits of Object Orientation

The most often stated benefits of object orientation:

- Code re-use
- Complexity management
- Easier maintenance
- Quicker time to market

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

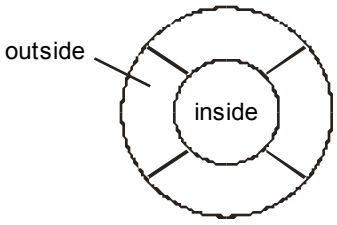
Goals of Object Orientation

- Reuse (the Holy Grail)
- Easier to understand code
- Malleability
- Easier to maintain and debug
- Manage complexity for large projects
- Model the real world

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is an Object?

- An object is a set of state and the behaviors that act on the state
- Objects have an "inside" and an "outside"



Object Name	Sample State	Sample Behavior
lassie	weight	bark
account12	balance	debit
policy765	amount	payOut

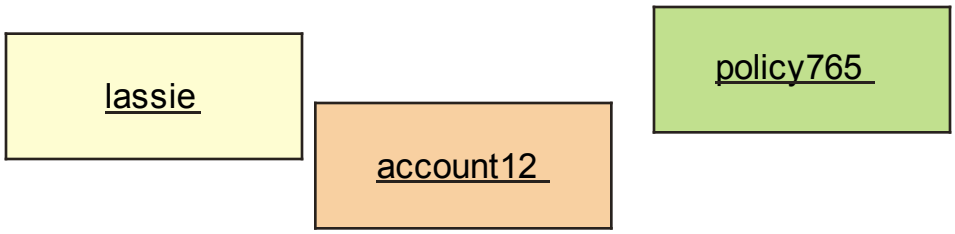
The "state" of an object refers to its data and the "behaviors" are the functions that operate on the data. In this course, we will use a "doughnut" diagram to show how objects have a protected "inside" and an accessible "outside".

The table on this page shows some sample objects and some of their states and behaviors. Note how the states are nouns, while the behaviors are verbs.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Symbols for Objects

- The UML symbol for an object is a box with the object's name underlined
- We also use a "doughnut" to represent objects to emphasize the inside/outside of the object



To represent objects in a diagram, UML uses a box with the object's name underlined. In this course, we will also use the non-standard doughnut symbol shown on the last page.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Thinking in Objects is Useful

- People naturally think of objects in the real world and intuitively understand the idea of state (e.g. a dog's weight) and behavior (e.g. eat)
- Using software objects like a "savings account" object in a software program minimizes the *representational gap* between the program and its real-world counterpart

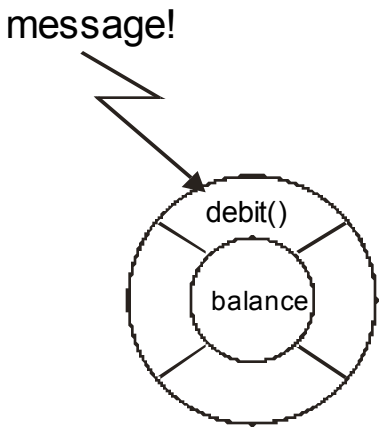
Humans are good at classifying real-world objects, and that skill carries over to OO programming. In fact, in some cases, it's harder for experienced programmers to learn OO techniques than it is for novices. That's because experienced programmers most likely will need to "unlearn" how to think like a computer and relearn how to naturally work with objects.

Minimizing the representational gap makes software easier to understand and helps to manage complexity.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Objects Respond to Messages

- We refer to executing an object's behavior as *sending a message* to the object
- The behaviors then typically act on, perhaps changing, the object's state



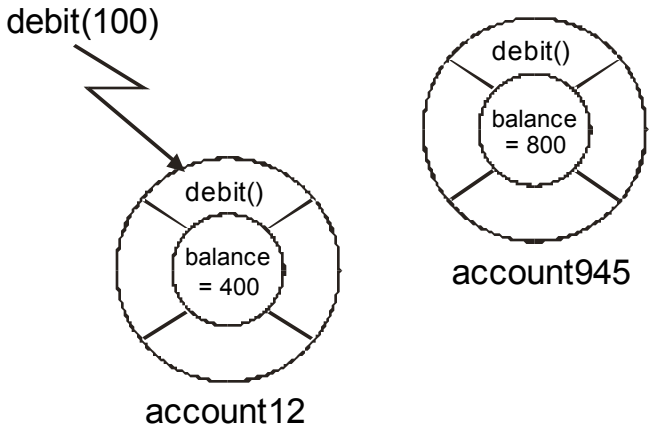
The notion of sending messages is central to object orientation. In other words, we think of objects as exposing behaviors that the outside world can invoke.

If you are coming from a traditional programming background, this might seem a bit strange, but it might help if you think of sending a message as a call to a function.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Each Object is Separate from All Others

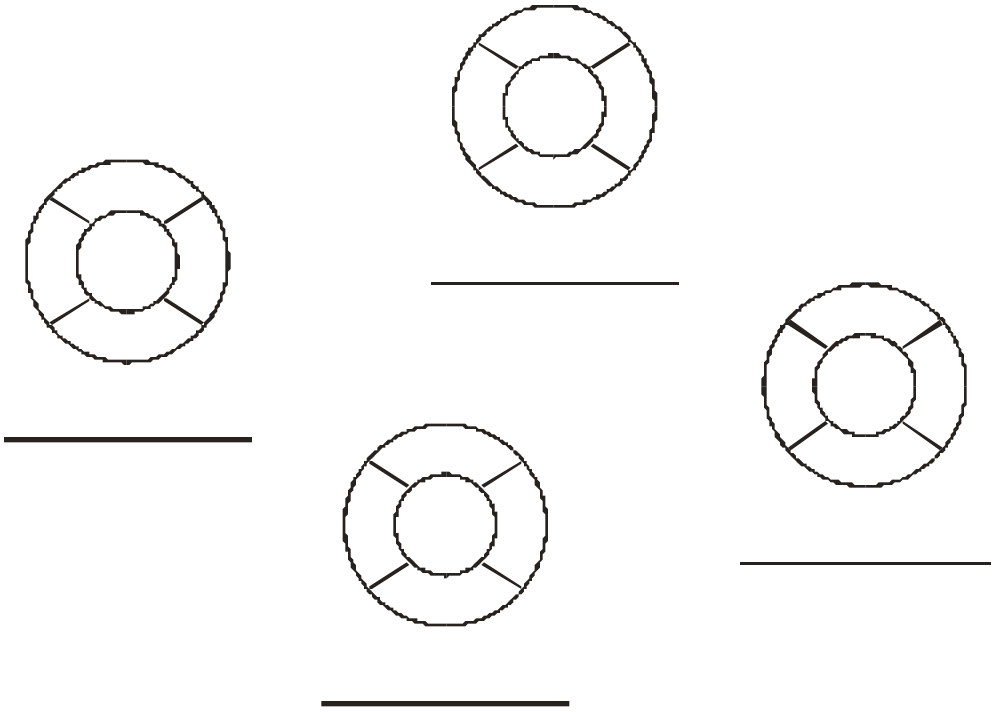
- Each object occupies a chunk of memory separate from all other objects
- The behaviors on an object typically act only on the state of the object receiving the message



Objects are self-contained, standalone entities and the act of sending a message to one object typically has no effect on other objects. Again, this models the real world. For example, if you withdraw money from one bank account, that act shouldn't change the balance of any other account.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sets of Similar Objects

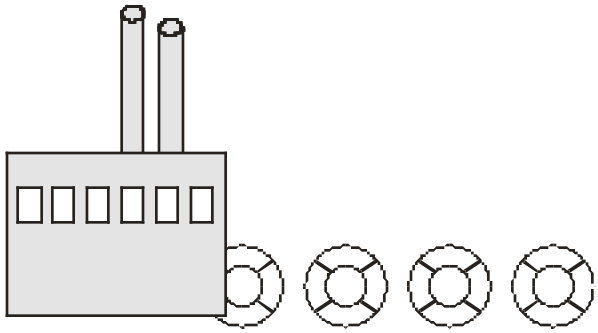


2 - 9

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Class is a Blueprint

- In most OO languages, you must define the class before you can create any objects
- The class acts as a template, or blueprint for creating the objects



2 - 10

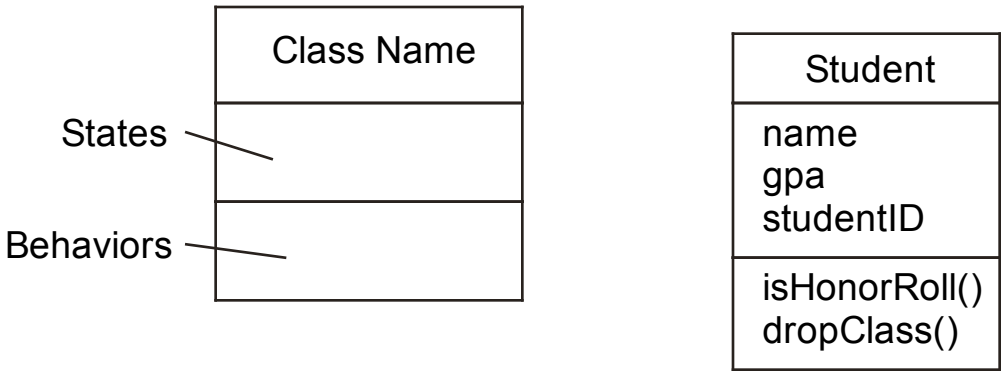
A class defines the states and behaviors of all of the objects in that class and then acts as a factory for creating objects.

In OOAD, we generally begin the analysis process by identifying objects. However, we get to the implementation state, most programming languages require us to define the class before we can create the objects.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

UML Symbol for a Class

- To represent a class, UML uses a box with compartments for the class name, state and behaviors



The class diagram is one of the central pieces of the UML -- we use class diagrams to model the classes in our software system and the relationships between the classes.

Note that the UML notation is not tied to any particular programming language. Also note that the behaviors have parenthesis in which we could specify arguments to the behaviors.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A PHP Class Definition

```
1    public class Student
2    {
3      private $studentID;
4      private $name;
5      private $gpa;
6
7      public function isHonorRoll()
8      {
9         if (gpa > 3.5) return true;
10        else return false;
11     }
12     . . .
13   }
```


Be sure that you can identify the states of this class (lines 3 to 5) and the behaviors. We didn't have enough room on this page to show all of the behaviors.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Creating Objects

- The act of using the class as a blueprint for creating objects is referred to as instantiation
- In PHP, you use the *new* keyword to instantiate objects

```
1    Student $sue = new Student();
2    Student $bill = new Student();
3         . . .
4    $hr = sue->isHonorRoll();
```


Again, don't worry too much about syntax, but note that lines 5 and 6 create a couple of Student objects and line 8 sends the "isHonorRoll()" message to one of the objects.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Three Pillars of Object Orientation

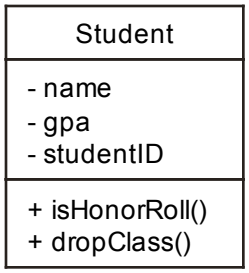
- Encapsulation
- Inheritance
- Polymorphism

Most object-oriented theorists consider these techniques to be the basic pillars of what it means to be object-oriented. We will overview each on the next few pages.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Encapsulation

- Encapsulation decreases the brittleness of classes by hiding implementation details -- the class developer can change the "inside" of the doughnut without worrying about outside side effects
- A good rule of thumb for getting started is to make all state private and behaviors public as needed
- UML uses access specifiers (+ or -) to indicate encapsulation



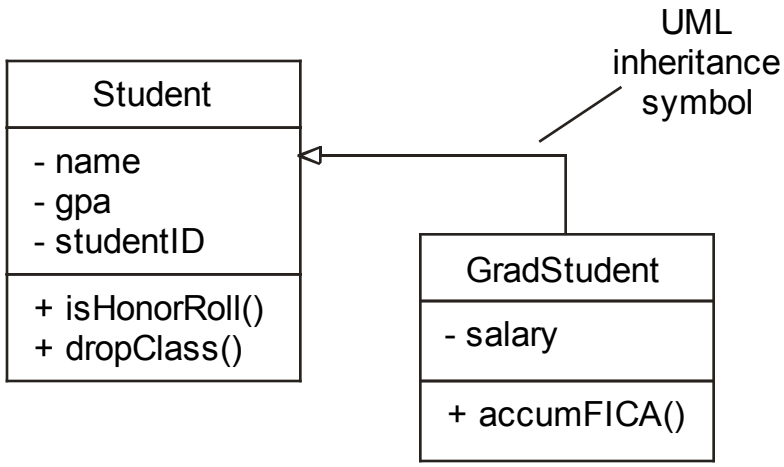
Encapsulation is a simple idea -- the "black box" principle. The idea is that if you hide the innards of a class, the outside world cannot depend on the innards -- that means that you can modify the innards without worrying about breaking other code.

Encapsulation reduces the brittleness of code because it lets you change (fix) parts of your program without breaking other parts. And since maintenance is often the largest expense in a software project, anything you can do to make the system more malleable will pay off.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Inheritance

- Inheritance is a strategy for re-using existing code -- you can create new classes that extend (supplement) existing classes
- UML uses a broad-headed arrowed line to show the relationship between the subclass and its superclass



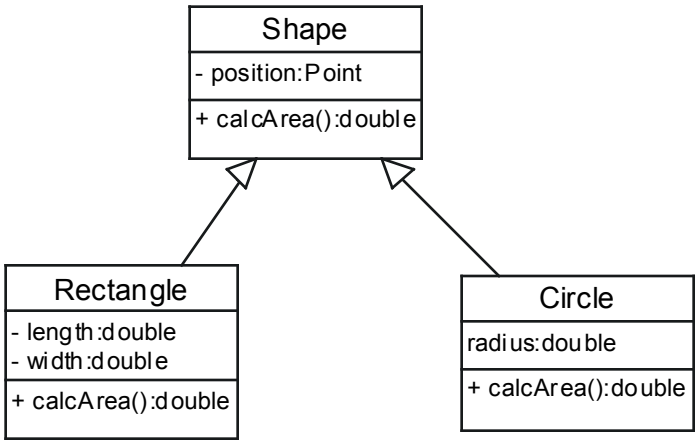
At its base, inheritance is a technique that lets use avoid rewriting code from scratch. The idea is that you can create re-useable classes and then "customize" those classes for a particular purpose by writing a subclass.

The subclass typically adds new states and/or behaviors and can also "override" behaviors from the superclass.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Polymorphism

- The word polymorphism means "many forms" and in conjunction with inheritance, lets you create malleable, extensible systems



Polymorphism is a tough concept to understand unless you also understand inheritance, so we will put off full coverage until later in the course. However, the basic idea is that a subclass can override behavior from the superclass and change the way the objects respond to messages.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Object Oriented Languages

- Various programming languages support varying degrees of object-oriented techniques
- Of course, it's best to use a language that fully supports all techniques

In theory, you can use any programming language to implement an object-oriented software system, but it will be a lot easier if the language natively supports the three pillars.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

PHP And O-O

- Basic object-oriented programming functionality was added in PHP 3 and improved in PHP 4
- Object handling was completely rewritten for PHP 5, expanding the feature set and enhancing performance

According to Wikipedia:

Prior to version 5, objects were handled like value types. The drawback of this method was that the whole object was copied when a variable was assigned or passed as a parameter to a method. In the new approach, objects are referenced by handle, and not by value.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Wegner Definitions

- Peter Wegner of Brown University devised a set of definitions that let us classify programming languages based on their OO support:

- *Object-based* languages support the notion of objects
- *Class-based* languages allow you to create classes to represent sets of objects
- *Object-Oriented* languages support inheritance and polymorphism

---

You can read the Wegner definitions at:

http://www.cse.msu.edu/~stire/cse891/wegner.pdf

Note that by these definitions, PHP is a true object-oriented language.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Review Questions

- What is the difference between an object and a class?
- Explain how encapsulation decreases the brittleness of a software system.
- Think about a software system for a bank. What sort of objects and classes might the system have?

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- Why thinking in objects is natural
- The UML symbols for classes and inheritance
- The Wegner Definitions

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum