Object Fundamentals
Part Two

Kenneth M. Anderson
University of Colorado, Boulder
CSCI 4448/6448 — Lecture 3 — 09/04/2007
Lecture Goals

- Continue our tour of the basic concepts, terminology, and notations for object-oriented analysis, design, and programming

- Some material for this lecture is drawn from *Head First Java* by Sierra & Bates, © O'Reilly, 2003
Overview

- Objects
- Classes
  - Relationships
    - Inheritance
    - Association
    - Aggregation/Composition
    - Qualification
  - Interfaces
Objects (I)

- OO Techniques view software systems as being composed of objects.
- Objects have:
  - **state** (aka attributes)
  - **behavior** (aka methods or services)
- We would like objects to be:
  - highly cohesive
    - have a single purpose; make use of all features
  - loosely coupled
    - be dependent on only a few other classes
Objects (II)

- Objects interact by sending messages to one another
  - A message is a request by object A to have object B perform a particular task
  - When the task is complete, B may pass a value back to A
- Note: sometimes A == B
  - that is, an object can send a message to itself
Objects (III)

- In response to a message, an object may
  - update its internal state
  - retrieve a value from its internal state
  - create a new object (or set of objects)
  - delegate part or all of the task to some other object

As a result, objects can be viewed as members of various object networks

- Objects in an object network (collaboration) work together to perform a task for their host application
Objects (IV)

- UML notation
  - Objects are drawn as rectangles with their names and types underlined
  - Ken : Person
  - The name of an object is optional. What is required is to list the object’s type
    - : Person
  - Note: the colon is not optional. It’s another clue that you are talking about an object, not a class
Objects (V)

- Objects that know about each other have lines drawn between them.

- This connection has many names, the three most common are:
  - object reference
  - reference
  - link

- Messages are sent across links.

- Links are instances of associations (see below).
Objects (Example)

Ken: Person

Skippy: Dog

Felix: Cat

sit()

feed()
Classes (I)

- A class is a blueprint for an object
  - The blueprint specifies the attributes (aka instance variables) and methods of the class
    - attributes are things an object of that class knows
    - methods are things an object of that class does
  - An object is instantiated (created) from the description provided by its class
    - Thus, objects are often called instances
Classes (II)

- An object of a class has its own values for the attributes of its class
  - For instance, two objects of the Person class can have different values for the name attribute
  - In general, each object shares the implementation of a class’s methods and thus behave similarly
- When a class is defined, its developer provides an implementation for each of its methods
- Thus, object A and B of type Person each share the same implementation of the sleep() method
Classes (III)

- Classes can define “class wide” (aka static) attributes and methods
  - A static attribute is shared among a class’s objects
  - A static method does not have to be accessed via an object; you invoke static methods directly on a class
  - We will see uses for static attributes and methods throughout the semester
Classes by Analogy

- Address Book
  - Each card in an address book is an “instance” or “object” of the `AddressBookCard` class
  - Each card has the same blank fields (attributes)
  - You can do similar things to each card
    - each card has the same set of methods
  - The number of cards in the book is an example of a static attribute; sorting the cards alphabetically is an example of a static method
Classes (IV)

- UML Notation
  - Classes appear as rectangles with multiple parts
    - The first part contains its name (defines a type)
    - The second part contains the class’s attributes
    - The third part contains the class’s methods

<table>
<thead>
<tr>
<th>Song</th>
</tr>
</thead>
<tbody>
<tr>
<td>artist</td>
</tr>
<tr>
<td>title</td>
</tr>
<tr>
<td>play()</td>
</tr>
</tbody>
</table>
Relationships: Inheritance

- Classes can be related in various ways
  - One class can extend another (aka inheritance)
    - notation: an open triangle points to the superclass
  - As we learned last time, the subclass can add behaviors or override existing ones
Relationships: Association

- One class can reference another (aka association)
- notation: straight line
- This notation is a graphical shorthand that one or both classes contain an attribute whose type is the other class
Multiplicity

- Associations can indicate the number of instances involved in the relationship
  - this is known as multiplicity
- An association with no markings is “one to one”
- An association can also indicate directionality
- Examples on next slide
### Multiplicity Examples

1. **One B with each A; one A with each B**
2. **Same as above**
3. **Zero or more Bs with each A; one A with each B**
4. **Zero or more Bs with each A; ditto As with each B**
5. **Two to Five Bs with each A; one A with each B**
6. **Zero or more Bs with each A; B knows nothing about A**
Associations can also convey semantic information about themselves.

In particular, aggregations indicate that one object contains a set of other objects. Think of it as a whole-part relationship between:

- A class representing a group of components
- A class representing the components

Notation: aggregation is indicated with a white diamond attached to the class playing the former role.
Example: Aggregation

Aggregation

Crate

Bottle

Composition

Book

Section

Chapter

Composition will be defined on the next slide
Semantics of Aggregation

- Aggregation relationships are transitive
  - if A contains B and B contains C, then A contains C
- Aggregation relationships are asymmetric
  - If A contains B, then B does not contain A
- A variant of aggregation is composition which adds the property of existence dependency
  - if A composes B, then if A is deleted, B is deleted
- Composition relationships are shown with a black diamond attached to the composing class
Relationships: Qualification

- An association can be qualified with information that indicates how objects on the other end of the association are found.
  - This allows a designer to indicate that the association requires a query mechanism of some sort.
  - E.g., an association between a phonebook and its entries might be qualified with a name, indicating that the name is required to locate a particular entry.
- Notation: a qualification is indicated with a rectangle attached to the end of an association indicating the attributes used in the query.
Qualification Example
A class can indicate that it implements an interface

- An interface is a type of class definition in which only method signatures are defined

A class implementing an interface provides method bodies for each defined method signature

- This allows a class to offer multiple types of services that are independent of its inheritance relationships

Other classes can then access a class via an interface

- This is indicated via a “ball and socket” notation
Example: Interfaces

Dog

- food type
- location

- makeNoise()
- eat()
- roam()

Pet

Dog

- food type
- location

- makeNoise()
- eat()
- roam()

Pet

Person
Class Summary

- Classes are blue prints used to create objects
- Classes can participate in multiple relationship types
  - inheritance
  - association
    - associations have multiplicity
  - aggregation/composition
  - qualification
  - interfaces
Coming Up Next

- Lecture 4: Object Fundamentals, Part 3
- Lecture 5: Great Software
- Read Chapter 1 of the OO A&D book