CS2704: Object-Oriented Software Design

Topic 2: Abstraction and Separation
Dr. Ben Keller
Fall 1999

Outline

- Abstraction
- Separation
- Identifying objects and classes

Abstraction

- General concept: describing an entity in terms of its aspects (qualities or features)
- Modeling entities in software
- Only essential aspects should be captured
  - Attributes
  - Behavior
- What is essential depends on situation

Example Abstraction

A named collection of attributes and behavior relevant to modeling a given entity for some particular purpose

A “Passenger” Abstraction

- Abstraction of a passenger for a flight reservation system
Attributes: Behaviors:
Properties of a Good Abstraction

- well named – clearly identifies abstraction
- coherent – sensible description
- accurate – only attributes of entity
- minimal – no irrelevant attributes
- complete – everything needed

Mapping Abstraction to Software

real-world abstraction software

entity

attributes {data, data, …}

behavior {method, method, …}

Mapping Abstraction to a Class

className

public

private

{data, data, …}

{method, method, …}

Separation

- Separation of what a component does from how it does it (ex. a procedure)
- Define classes by independently specifying the interface for objects in that class, and the implementations of that interface

Separation of Interface and Implementation

Interface

Implementation

visible

hidden

Interchangeability

Implementation 1

Implementation 2

Interface
Separation of Classes

- Provides method
- Identifies available methods
- Uses method

General Structure of a Class

className

public

private

{data, data, ….}

{method, method, ….}

General Structure of an Object

Multiple Instances of a Class

Identifying Objects and Classes

- Study features of system
- Look for nouns (people, places, things)
- Example features:
  - “add course grade to student record”
  - “enter rental equipment description”
  - “add frequent flyer miles to customer record”

Good Classes

- Class should represent a set of objects although sometimes only use one
- Behaviors (methods) of class should be meaningful
  - Should a chess piece move itself?
  - What behaviors does a piece of “Data” have?
Discovering and Rejecting Classes

- B. Meyer, Object-Oriented Software Construction, 2nd Ed, Prentice-Hall, 1997
- Meyer is author of Eiffel language/method

Sources of Class Ideas

<table>
<thead>
<tr>
<th>Source of Ideas</th>
<th>What to look for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing Libraries</td>
<td>•Classes that meet needs of application</td>
</tr>
<tr>
<td>Requirements Documents</td>
<td>•Terms that occur frequently</td>
</tr>
<tr>
<td>Requirements Documents</td>
<td>•Terms with explicit definitions</td>
</tr>
<tr>
<td>Requirements Documents</td>
<td>•Terms with assumed meaning</td>
</tr>
</tbody>
</table>

Sources of Class Ideas (2)

<table>
<thead>
<tr>
<th>Source of Ideas</th>
<th>What to look for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discussion with customers and users</td>
<td>•Important abstractions in the application domain</td>
</tr>
<tr>
<td>Discussion with customers and users</td>
<td>•Jargon of application domain</td>
</tr>
<tr>
<td>Discussion with customers and users</td>
<td>•Both conceptual and material objects</td>
</tr>
<tr>
<td>Documentation for similar systems in same domain</td>
<td>•Useful design abstractions</td>
</tr>
<tr>
<td>Documentation for similar systems in same domain</td>
<td>•See above</td>
</tr>
</tbody>
</table>

Sources of Class Ideas (3)

<table>
<thead>
<tr>
<th>Source of Ideas</th>
<th>What to look for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-O-O systems or system descriptions</td>
<td>•Data elements passed as arguments</td>
</tr>
<tr>
<td>Non-O-O systems or system descriptions</td>
<td>•Shared memory areas</td>
</tr>
<tr>
<td>Non-O-O systems or system descriptions</td>
<td>•Important files</td>
</tr>
<tr>
<td>Non-O-O systems or system descriptions</td>
<td>•Records (or structs) used by several procedures of modules</td>
</tr>
<tr>
<td>Non-O-O systems or system descriptions</td>
<td>•Entities in ER modeling (DB)</td>
</tr>
</tbody>
</table>

Sources of Class Ideas (4)

<table>
<thead>
<tr>
<th>Source of Ideas</th>
<th>What to look for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discussions with experienced designers</td>
<td>•Design classes used successfully in other similar systems</td>
</tr>
<tr>
<td>Algorithms and Data Structures literature</td>
<td>•Known data structures with efficient operations</td>
</tr>
<tr>
<td>OO design literature</td>
<td>•Relevant design patterns</td>
</tr>
</tbody>
</table>

Rejecting Classes

<table>
<thead>
<tr>
<th>Danger Sign</th>
<th>Suspicions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class with verbal name (infinitive or imperative)</td>
<td>May be a procedure, not a class</td>
</tr>
<tr>
<td>Class with only one public method</td>
<td>May be a procedure, not a class</td>
</tr>
<tr>
<td>Class described as “performing” something</td>
<td>May not be a proper data abstraction</td>
</tr>
</tbody>
</table>
## Rejecting Classes (2)

<table>
<thead>
<tr>
<th>Danger Sign</th>
<th>Suspicion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class with no methods</td>
<td>May not be an ADT, or missed methods</td>
</tr>
<tr>
<td>Class involving several abstractions</td>
<td>Should be split into one class per abstraction</td>
</tr>
<tr>
<td>Class that inherits from another class, but has few features itself</td>
<td>A case of taxonomic overkill</td>
</tr>
</tbody>
</table>