Designing the Classes

- Once a set of candidate objects is determined... we must:
  - Determine which are "real" objects in the system.
- Identify their attributes:
  - attributes are data
  - define what the data is, not how it is to be represented (that comes later)
- Identify their responsibilities:
  - public services (behaviors) the object must provide
  - may imply certain attributes necessary to provide those services
  - define what the service is, not how it is to be accomplished
  - some services may be private, but those are usually identified later
  - services are invoked through message passing

Identifying Attributes

- An attribute is a single characteristic which is common to all instances of a class.
- Look for adjectives and possessive phrases in the requirements document.
- Find a general description of the object.
- Determine what parts of the description are applicable to the problem domain.
- Four categories of attributes:
  - descriptive
  - naming
  - state information
  - referential (relationship links)

Eliminating Attributes

- Some apparent attributes may be considered independently of the objects — make those objects in their own right.
  - Rumbaugh: if an attribute is changed in the system w/o being part of any entity, then it should be an object.
- Relationships among objects may also have attributes. Do not confuse those with attributes of the involved objects.
- Eliminate minor details that do not affect methods.

Specifying Attributes

- An attribute should be atomic (simple).
- Eliminate attributes that can be calculated from others.
- Eliminate attributes that address normalization, performance, or other implementation issues.
- Verify that the attributes make semantic sense together.
Data Versus State

<table>
<thead>
<tr>
<th>Data</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition:</strong></td>
<td><strong>Definition:</strong></td>
</tr>
<tr>
<td>Information processed by</td>
<td>Information used by system</td>
</tr>
<tr>
<td>the system</td>
<td>to control processing</td>
</tr>
<tr>
<td><strong>Example:</strong></td>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td>The identity of the</td>
<td>Whether a book is already</td>
</tr>
<tr>
<td>patron and the book</td>
<td>checked out, available,</td>
</tr>
<tr>
<td>when a book is</td>
<td>or unknown when a request</td>
</tr>
<tr>
<td>checked out to the</td>
<td>is made for it.</td>
</tr>
<tr>
<td>patron</td>
<td>Perhaps the number of</td>
</tr>
<tr>
<td></td>
<td>books checked</td>
</tr>
<tr>
<td></td>
<td>out to a patron.</td>
</tr>
</tbody>
</table>

Identifying Responsibilities

Look for verb in the requirements document — usually this will define services of the object of the sentence.

E.g. Quarterback throws the ball.
This defines a service for the ball, provided by the quarterback.

Look at user scenarios — different ways the system can be used.

Look at each feature — require services of many objects.

Specifying Responsibilities

Name the service to match the external request for the service.
- RegisterPatron
- CheckInBook

Identify the information and/or entities necessary to provide the service.
- patron name, address, patron list
- book ISBN or call number, patron id, catalog, patron list, ??

Identify the responses, if any, that the service will generate.
- success, failure, patron id
- success, invalid call number, invalid patron

Example: Library System

Consider the Patron class for the Library System:

- **Name:** Patron
- **Attributes:**
  - Name, Address, Membership Number
  - Fees due??
  - List of checked out items??
- **Responsibilities:**
  - Report Name, Address, Membership Number
  - Update fees due??
  - Record books this patron has checked out??

When are the attributes set?
Which of the attributes are mutable?
Guidelines for Designing the Classes

We need a systematic way of determining the attributes and responsibilities of a class.

Otherwise, we run a large risk of missing essential features.

To identify attributes and responsibilities the designer must ask the right questions regarding the system being designed.

We can provide some guidance in choosing what questions to ask…

Design Perspectives

Behavioral
Emphasizes actions in system

Informational
Emphasizes role of information/data/state and how it’s manipulated

Structural
Emphasizes relationships among components

Example: Library System

Specification:
Design a library catalog system. The system must support the registration of patrons, adding and removing books, checking books in and out, satisfying queries regarding the availability of books, and determining which patron has a book.

Behavioral (actions):
- patrons are registered (who does this??)
- books are checked in/out (who does this??)

Structural (relationships):
- catalog contains a list of books
- patron may have one or more books
- someone has a list of patrons

Informational (state):
- a book may be available/checked out??
- a book may be checked out to a specific patron (state or relationship??)

Behavioral Perspective

Consider some action in a program…

What object…
- initiates action?

What objects…
- help perform action?
- are changed by action?
- are interrogated during action?

Consider registering a patron…

Controller (procedural)…
- initiates the action

Circulation Desk…
- performs the action

Patron List…
- is changed by the action
- is interrogated during the action
**Behavioral Categories**

- **Actor** (does something)
  - Circulation Desk

- **Reactor** (system events, external & user events)
  - Controller, Parser??

- **Agent** (messenger, server, finder, communicator)
  - Catalog, PatronList

- **Transformer** (data formatter, data filter)
  - Parser

**Structural Perspective**

- What objects...
  - are involved in relationship?
  - are necessary to sustain (implement, realize, maintain) relationship?

- What objects not in relationship...
  - are aware of and exploit relationship?

Consider a relationship: book is checked out to patron

- Circulation Desk... is involved in establishing the relationship
- Catalog and PatronList... are necessary to sustain the relationship
- ??... is aware of and exploits the relationship

**Structural Categories**

- **Acquaintance** (symmetric, asymmetric)
  - CirculationDesk knows about Catalog, asymmetric relationship

- **Containment** (collaborator, controller)
  - CirculationDesk controls/uses CheckedOut

- **Collection** (peer, iterator, coordinator)
  - PatronList contains and manages Patrons
  - Catalog contains and manages Books
  - CheckedOut contains and manages ??

**Informational Perspective**

- What objects...
  - represent the data or state?
  - read data or interrogate state?
  - write data or update state?

Consider a state: status of book

- CheckedOut list, PatronList and Catalog implicitly... represent (stores) the state information
- CirculationDesk... interrogates the state of a book (via ?)
- CirculationDesk... updates the state of a book
Example: Preliminary Overall Design

Here's a partial, preliminary design, based on the preceding discussions:

- **Catalog**
- **Circulation**
- **Patron List**

- **Book**
  - string Author
  - string Title
  - string Call

- **CheckedOut**
  - string Call
  - string PatronID

**Patron**

means "contains a collection of"

means "knows about"

For simplicity, this omits the procedural controller and the parser.

Example: Library System

Consider the Patron class for the Library System:

- **Name:** Book
- **Attributes:**
  - Title
  - Author
  - Call Number

- **Responsibilities:**
  - Report Title
  - Report Author
  - Report Call Number

All of these attributes are immutable.

Provide access, but not modification.

Consider the Patron class for the Library System:

- **Name:** CirculationDesk

**Attributes:**
- Catalog
- PatronList
- CheckedOutList

- **Responsibilities:**
  - RegisterPatron
  - CheckBookIn
  - CheckBookOut
  - SearchByCallNumber

Each of these attributes is another object...

...but these may be links (pointers or references).

Provide support for the basic operations from the spec through the CirculationDesk interface.