

Designing the Classes

Designing Classes 1

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 to be accomplished
- some services may be private, but those are usually identified later
- services are invoked though message passing

Identifying Attributes

Designing Classes 2

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

Designing Classes 3

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

Designing Classes 4

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

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.

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

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?

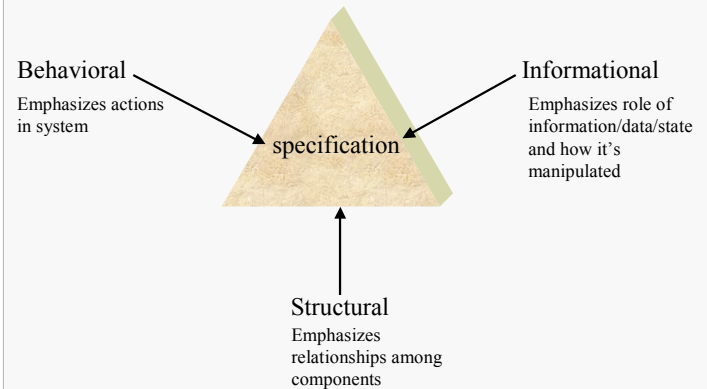
Which of the attributes are mutable?

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...



Kafura

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??)

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

Patron List...

- is interrogated during the action

Behavioral Categories

Designing Classes 13

- 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

Designing Classes 14

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

Designing Classes 15

- 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

Designing Classes 16

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 ...)

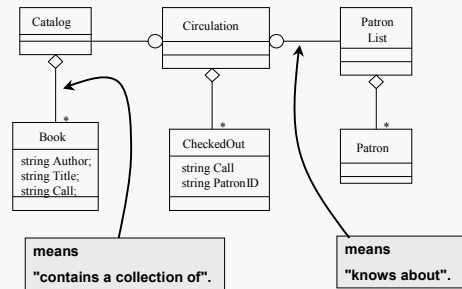
CirculationDesk...

- updates the state of a book

Example: Preliminary Overall Design

Designing Classes 17

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

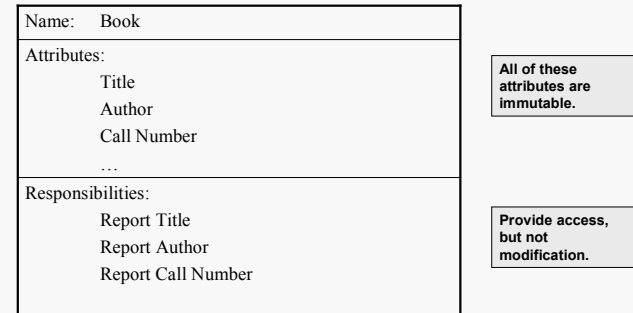


For simplicity, this omits the procedural controller and the parser.

Example: Library System

Designing Classes 18

Consider the Patron class for the Library System:



Example: Library System

Designing Classes 19

Consider the Patron class for the Library System:

