Identifying Objects and Classes

We must:

- identify potential objects from the specification
- eliminate phony candidates
- determine how the legitimate objects will interact
- extrapolate classes from the objects

This process:

- requires experience to do really well
- requires guidelines, none of which is entirely adequate
- often uses several approaches together
- should lead to too many rather than too few potential objects

Identifying Objects and Classes

Abbott and Booch suggest:

- use nouns, pronouns, noun phrases to identify objects and classes
- singular → object, plural → class
- not all nouns are really going to relate to objects

Coad and Yourdon suggest:

- identify individual or group "things" in the system/problem

Ross suggests common object categories:

- people
- places
- things
- organizations
- concepts
- events

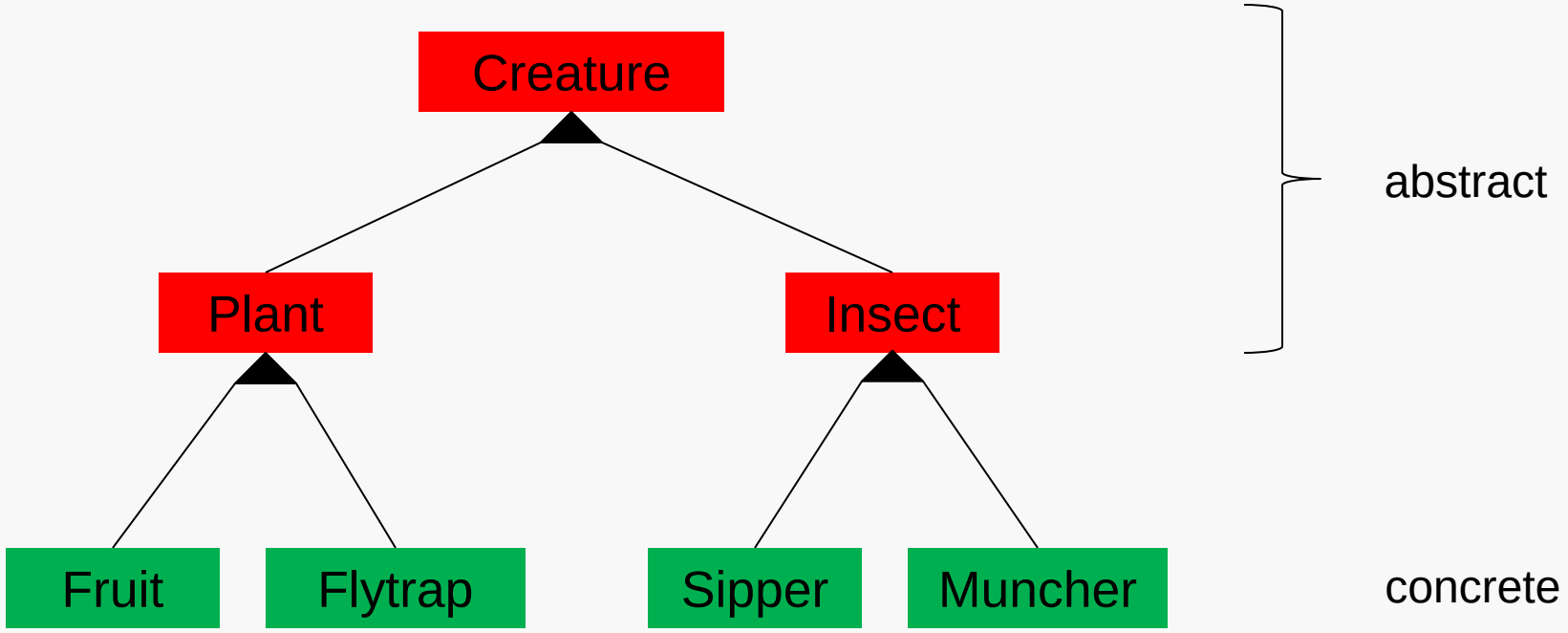
Example: P. E. S. T.

A little Abbott and Booch (nouns, noun phrases):

creatures
clock
ecosystem
plants, insects (kinds of creatures)
grid
script file
commands
name (of creature)
energy level
location
sipper, muncher (kinds of insects)
fruit, flytrap (kinds of plants)

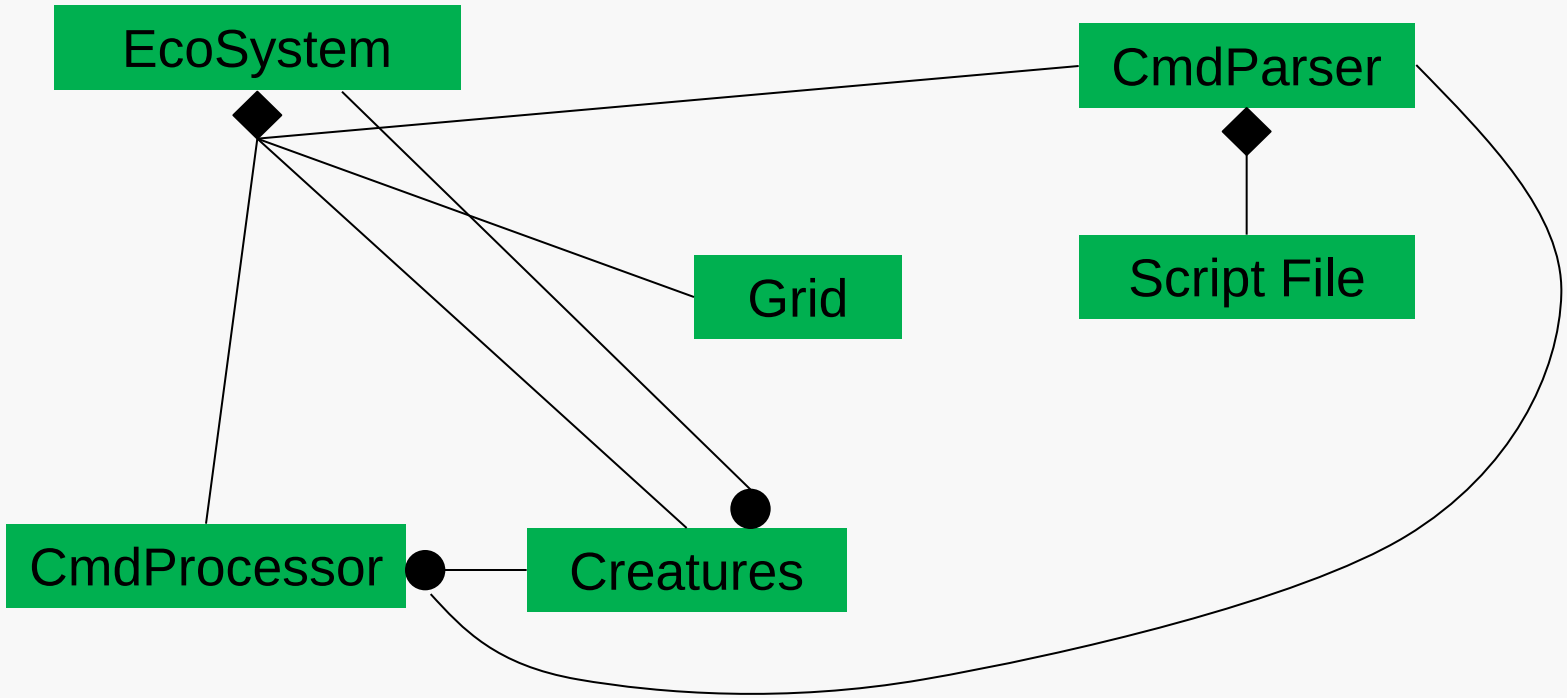
Example: P. E. S. T.

Some consideration of the specification reveals some is-a-kind-of relationships:



Example: P. E. S. T.

More reflection (including a bit of Coad and Yourdon), suggest some higher-level organizational details:



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

Data	State
Definition: Information processed by the system	Definition: Information used by system to control processing
Examples from Minor 1: a record offset a GIS record	Examples from Minor 1: type of current command

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 components 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.

- reportFID()
- serveNextCommand()
- getRecordAtOffset()

Identify the information and/or entities necessary to provide the service.

- GIS record object
- command file, command file processor

Identify the responses, if any, that the service will generate.

- feature ID (cannot fail unless object not initialized)
- no more commands in file
- invalid file offset

Example: P. E. S. T.

Consider the Sipper and Muncher classes:

What do they have in common?

- movement
- uses energy when moving
- feeding
- have energy level

. . .

How do they differ?

- movement pattern
- energy usage when moving
- feeding pattern
- muncher divides in two

. . .

What do these facts imply about the Sipper and Muncher classes?

What do these facts imply about the Insect class?

Guidelines for Designing the Classes

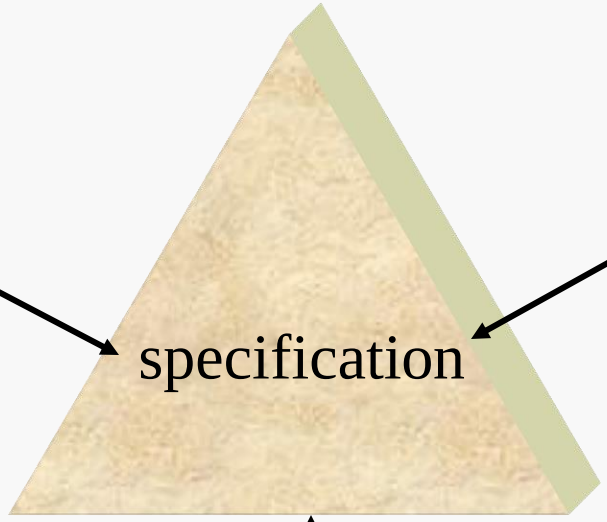
We need a systematic way of determining the attributes and responsibilities of a class.

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

Kafura

Example: P. E. S. T.

Consider the Muncher class:

Behavioral perspective (actions):

- updates state (energy level, location) on clock tick (notified of that by whom?)
- detects presence of edible plant when moving (how?)
- feeds (receives energy from plant (how?)
- breeds (what do "children" receive? how does system know about new Munchers?

Structural perspective (relationships):

- must be able to "know about" plant it's feeding on
- must "know about" grid to detect presence of other creatures

Informational perspective (state):

- location
- energy level

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?

Behavioral Categories

Actor (does something, typically initiates)
Controller

Reactor (system events, external & user events)
Controller, CommandProcessor (?)

Agent (messenger, server, finder, communicator)
possibly CommandParser, GISRecordFileparser

Transformer (data formatter, data filter)
possible CommandParser, GISRecordFileParser

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?

Structural Categories

Acquaintance (symmetric, asymmetric)

- Controller knows about CommandProcessor, asymmetric relationship

Containment (collaborator, controller)

- GISRecordFileParser controls/uses RandomAccessFile
- similar issue with CommandParser

Collection (peer, iterator, coordinator)

- Controller knows and manages CommandParser and CommandProcessor
- no data structrures issues as yet, but they would qualify

Informational Perspective

What objects...

- represent the data or state?
- read data or interrogate state?
- write data or update state?