# 6.005 Elements of Software Construction | Fall 2008 Problem Set 3: The Relational Paradigm

The purpose of this problem set is to give you practice in the basic techniques of the relational paradigm. You'll construct a variety of object models, and code one of them using standard patterns. You'll also do a small exercise to check that you understand some of the complexities of of mutable datatypes.

## Mutable Datatypes

**Investigating Java's Set**. The specification of `java.util.Set` says that a set should not contain itself. Write a JUnit test for one of the standard Java library implementations of `Set` that violates this rule and fails; explain what's happening carefully and succinctly; and explain how the principles of the lecture on mutable types address this issue.

## Object Modeling

For each of the following problems, construct (1) a graphical object model, accompanied by (2) designations that give a crisp and precise one sentence meaning for each set and relation name, and (3) three object diagrams showing interesting cases of instances satisfying the model.

**Sudoku**. Model the state of a puzzle, allowing for both completed and incomplete grids, and making explicit the grouping into rows, columns and subgrids.

**Slide Animation**. Model a slide containing textual and graphical elements, along with settings for animation. Include grouping of elements, and appearance order and effects.

**Domain Name System (DNS)**. Model domain names and the architecture for resolving them. Your model should include the structure of the domain names, their relationship to IP addresses, the network of domain name solvers, and caches (and little else!). For an explanation of DNS, see the wikipedia article. Do not worry about the low level structure of DNS records; your model should show the structure of the system (and the state of the servers) at an abstract level.

**Social Network**. Model a social network (as found, for example, in Facebook, MySpace and LinkedIn) that includes the following features: members, friendship, groups, profiles, posted objects, tagged photos, recent activity feeds, pending invitations.

**Election Ballots**. Model the voter input in a US federal election, consisting of a set of marked ballots, containing sufficient information to determine the outcome of the presidential election, senate and congressional races, and state ballot questions.

**Address Book**. Model an email address book that associates user-defined names with email addresses, and includes the ability to distinguish home and work addresses, forming groups of multiple addresses, and using indirection to allow groups to contain groups and user-defined names.

## Object Model Implementation

Implement an address book, based on the object model that you constructed. Your address book should provide an API, as a class `AddressBook`, that provides key operations for entering and editing entries, displaying entries, and for resolving a user-defined name to the email addresses it represents. Indicate precisely which object model implementation patterns you used, and how they were applied to derive the implementation structure from the object model. Construct a suite of test cases based on partitioning that gives reasonable coverage of the specification of the API, and make sure to include the three sample instances that you devised for your object model.

Your solution will be judged on the clarity of structure of your code, on how well you mapped the model to the code, on the quality of design of your API, and on quality of the testing you performed.

## Infrastructure

No code is provided for this problem set. A directory called ps3 will be created for you in your personal repository, containing a copy of this file. In addition to your code and test cases, you should commit your solutions to the exercises as a single PDF file.