Objects and Classes

I. What are Objects and Classes?

A. What are Objects?

1. [Booch] defines an object as something that "has state, behavior, and identity."
2. Examples: An electronic mail system: mailboxes, messages, passwords, menus
3. Three properties of objects:
   a. State: Can store information that is result of prior operations, which has an affect on future operations
      E.g. mailboxes: state (empty, full) affects whether respond or print are valid operations.
   b. Operations: Can add messages or print them, but can’t edit received messages
   c. Identity: Even though there is one "mailbox" class, there could be many different instances of the class. E.g. Each person has one or more mailboxes
4. There is also such a thing as process or event objects, e.g. which might represent the real-world mail handler process(es), or a process that responds to voice commands

B. What are Classes?

A class describes a collection of related objects. Similar objects are grouped into classes.
1. A class description lists:
   a. Possible states for objects of the class (e.g. mailbox full, empty) stored in variables
   b. The operations allowed on the objects of the class (e.g. add a message, delete a message). These are functions (aka methods) associated with objects of the class.
2. Objects that conform to a class description are called instances of that class.
   E.g. a schematic of the mailbox class and two of its instances

C. Inheritance: Allows exploiting similarities between classes, giving rise to a class hierarchy

E.g. some messages (a subset of all of them) include graphics or sound.
1. A "SoundMessage" class would be a subclass of the "Message" class
2. The "Message" class is a superclass (a.k.a. parent or base class) of the "SoundMessage" class
3. Note that a subclass is actually a superset, as it includes everything from the parent with the addition of new information
4. A subclass object must be legally usable in all situations in which a superclass object is expected (e.g. can delete, store, or forward a "SoundMessage")
5. Consider messaging (method look-up) under inheritance: follows hierarchy
6. Question: Consider checking & savings accounts in a bank. How do they differ? Should they be separate classes, or instances of the same class? Should they inherit from a shared parent class called Account? How do they differ from Account?
II. The Development Process (From Problem to Code)

The development process can be loosely broken down into Analysis, Design, and Implementation.

A. Analysis Phase:
   1. Yields an analysis document that is a combination of a user manual and a reference manual.
   2. Concentrates on the *what* rather than *how*.
   3. E.g. Think of creating a word-processing program. Fonts, footnotes, headers, etc. must be defined.
   4. This document is:
      a. readable both by experts in the problem domain and by software developers
      b. Free from internal contradictions
      c. A complete definition of the tasks to be solved
   5. *Question*: What would be entailed in the analysis document for a simulated bank teller machine? It would need to be specific enough to be a legal contract before the work was done.

B. Design Phase
   1. Structure the programming tasks into *classes* and *class clusters*
   2. List each classes operations and relationship to other classes
   3. Ignore exact choice of data structures and choice of programming language
   4. *Question*: How would the design differ from the analysis for the bank teller problem?

C. Implementation Phase
   1. Traditionally this is integration of procedural units which have been developed separately
   2. Gradual refinement of a prototype works well in a OOP language, since prototype class are likely to exist in final version as well.

III. Issues in Object-Oriented Development

A. Classes and Operations
   1. Consider what we can do with a mailbox object: Add, play, or delete a message.
   2. How about operations on a message object itself? -Play it

B. The Object-Oriented Design Process
   1. Three goals
      a. Identify the classes
      b. Identify the functionality of these classes
      c. Identify the relationships between these classes
   2. The above are goals, not steps

C. Finding Classes
   1. Look for *nouns* in the problem analysis
   2. Examples: Mailbox, Message, User, Password, SuperUser, MailSystem, Menu
      Need for other classes may arise, e.g. MessageQueue class handling messages in FIFO (can ignore underlying implementation for now)

D. Finding Operations
   1. Look for *verbs* in problem description
   2. Examples: messages can be recorded, played, deleted
   3. MessageQueue class operations: Initialize, add a message, remove a message, test if empty
   4. Some combinations of class & operation are poor.
      E.g. We have operation *play* a message, but should we have operation *add* a message to a mailbox? No - we would need to know internal structure of mailbox.
   5. We assume that an object has no insight into the internal structure of another object
6. Question: Consider page layout in a word processor. Is a Page object responsible for fitting Paragraph and Figure objects inside, or should it be the other way around? What internal structure knowledge is required in either case?

E. Finding Class Relationships

Three relationships are common: Use or awareness, Aggregation or containment, and Inheritance or specialization

1. Use or Awareness:
   a. Class A uses class B if
      (1). An operation of A receives or returns an object of class B
      (2). In process of an operation of A, an object of class B must be inspected or created
      (3). Objects of A contain objects of class B or references to them
   b. If A can carry out all its tasks without being aware that B exists, then it does not use B. (e.g. Message does not use Mailbox)
   c. We want to minimize coupling

2. Aggregation or containment:
   "has-a" relationship. E.g. class A contains objects of class B, such as a Mailbox has a Message. Class A must be aware of class B

3. Inheritance or specialization
   "is-a" relationship.
   a. Class A inherits from class B if all objects of class A are also objects of class B.
   b. All B operations must be valid for A objects (though implementation may differ)
   c. E.g. sysadmin (root) is a user, ostrich is a bird

4. Question: Consider the classes discovered in the previously discussed automated teller design problem. Find the relationships between its classes. List them or draw diagrams representing their relationships. Alternatively do this same exercise with the Mail messaging or word processing examples.

IV. Design Hints

A. Classes Model Sets of Objects

A class is a set, and so should not be used to model a single object. For example, consider the following problem representing an acceptor of alphabetic strings ending in ab:

1. We can describe a class State and an operation Transition, with four derived classes Sinit, S1, S2, S3.
2. Alternatively we can have a class State with a lookup table containing all valid transitions. We would also need an operation to set table entries. This could then work for a general finite-state machine.

B. Classes Need Meaningful Operations

Avoid over-generalization in a design. Only develop the classes needed. If you have no idea what the operations are of your "general" classes, then they are too general.