1. For a statistical project you need to compute lots of random numbers in various ranges. (Recall that (random 10) returns a random number between 0 and 9, and more generally (random n) returns a random number between 0 and n.) Also, you need to keep track of how many random numbers are computed in each range. You decide to use object-oriented programming. Objects of the class random-generator will accept two messages. The message number means “give me a random number in your range” while count means “how many number requests have you had?” The class has an instantiation argument that specifies the range of random numbers for this object, so

(define r10 (instantiate random-generator 10))

will create an object such that (ask r10 ‘number) will return a random number between 0 and 9, while (ask r10 ‘count) will return the number of random numbers r10 has created.

2. Define the class coke-machine. The instantiation arguments for a coke-machine are the number of Cokes that can fit in the machine and the price (in cents) of a Coke:

(define my-machine (instantiate coke-machine 80 70))

creates a machine that can hold 80 Cokes and will sell them for 70 cents each. The machine is initially empty. Coke-machine objects must accept the following messages:

Continued on next page.
Continued...

(ask my-machine 'deposit 25) means deposit 25 cents. You can deposit several coins and the machine should remember the total.

(ask my-machine 'coke) means push the button for a Coke. This either gives a Not enough money or Machine empty error message or returns the amount of change you get.

(ask my-machine 'fill 60) means add 60 Cokes to the machine.

Here’s an example:

(ask my-machine 'fill 60)
(ask my-machine 'deposit 25)
(ask my-machine 'coke)

NOT ENOUGH MONEY

(ask my-machine 'deposit 25) ;; Now there’s 50 cents in there.
(ask my-machine 'deposit 25) ;; Now there’s 75 cents.
(ask my-machine 'coke) ;; return val is 5 cents change.

You may assume that the machine has an infinite supply of change.

3. We are going to use objects to represent decks of cards. You are given the list ordered-deck containing 52 cards in standard order:

(define ordered-deck '(AH 2H 3H ... QH KH AS 2S ... QC KC))

You are also given a function to shuffle the elements of a list:

(define (shuffle deck)
  (if (null? deck)
      '()
      (let ((card (nth (random (length deck)) deck)))
        (cons card (shuffle (remove card deck))))))

A deck object responds to two messages: deal and empty?. It responds to deal by returning the top card of the deck, after removing that card from the deck; if the deck is empty, it responds to deal by returning (). It responds to empty? by returning #t or #f, according to whether all cards have been dealt.

Write a class definition for deck. When instantiated, a deck object should contain a shuffled deck of 52 cards.

Continued on next page.
Continued ...

4. We want to promote politeness among our objects. Write a class \texttt{miss-manners} that takes an object as its instantiation argument. The new \texttt{miss-manners} object should accept only one message, namely \texttt{please}. The arguments to the \texttt{please} message should be, first, a message understood by the original object, and second, an argument to that message. (Assume that all messages to the original object require exactly one additional argument.) Here is an example using the \texttt{person} class from the upcoming adventure game project:

\begin{verbatim}
> (define BH (instantiate person 'Brian BH-office))
> (ask BH 'go 'down)
BRIAN MOVED FROM BH-OFFICE TO SODA

> (define fussy-BH (instantiate miss-manners BH))
> (ask fussy-BH 'go 'east)
ERROR: NO METHOD GO

> (ask fussy-BH 'please 'go 'east)
BRIAN MOVED FROM SODA TO PSL
\end{verbatim}