(For a more detailed presentation of this material, see Part X, Flavors, in volume 2 of the Symbolics documentation. I have skipped many features of flavors which you may find useful, and which are fully described there.)

The flavor system is the lisp machine's mechanism for defining and creating active objects, that is, objects which can receive messages and act on them. A flavor is a class of active objects. One such object is called an instance of that flavor.

There are two primary characteristics of a flavor: the set of messages an instance of that flavor can receive, and the set of state variables an instance of that flavor has. The state variables are called instance variables. Every object of a given flavor has the same set of instance variables, but the values of those instance variables are likely to vary from object to object. And for each message a flavor can receive, it has a corresponding function to invoke. The function which gets called to handle a particular message is called the flavor's method for that message. That method is shared by all instances of the flavor.

So, for instance, the window you see on a freshly booted machine is an instance of the flavor tv:lisp-listener. Like any instance of lisp-listener, it can handle 286
different messages (as of the current software). One of the messages it handles is :expose. Its expose method is a function which makes the lisp-listener visible on your screen, if it is not already. All lisp-listeners have the same expose method. One of the instance variables of flavor lisp-listener is exposed-p. All lisp-listeners have an exposed-p instance variable. If a given lisp-listener happens to be exposed, perhaps because you just sent it the :expose message, the value of its exposed-p instance variable will be t. Otherwise it will be nil.

2.1 Basic Usage

Flavors are defined with the defflavor special form. Here is a simple definition of a flavor named "ship," which might be used in a program for a space wars game.

```
(defflavor ship
  (x-position y-position
   x-velocity y-velocity mass)
  () )
```

It states that all instances of flavor ship will have five instance variables, as listed. (The empty list following the instance variables is related to a feature we'll consider in the section "Mixing Flavors.") Here are two methods for the ship flavor, to handle the messages :speed and :direction.

```
(defmethod (ship :speed) ()
  (sqrt (+ (expt x-velocity 2)
           (expt y-velocity 2))))

(defmethod (ship :direction) ()
  (atan y-velocity x-velocity))
```

A defmethod looks very much like a defun. It has a function-spec, an argument list, and a body. The body will be executed in an environment in which the names of ship's instance variables will refer to the instance variables of the specific ship object which received the message.

We might also wish to have methods which allow one to examine the values of ship's instance variables. Like:

```
(defmethod (ship :x-position) ()
  x-position)
```