LISP - CLOS

Common LISP predated the advance of object-oriented programming by couple of decades. However, it object-orientation was incorporated into it at a later stage.

Defining Classes

The **defclass** macro allows creating user-defined classes. It establishes a class as a data type. It has the following syntax:

```
(defclass class-name (superclass-name*)
   (slot-description*)
   class-option*))
```

The slots are variables that store data, or fields.

A slot-description has the form $slot - nameslot - option*$, where each option is a keyword followed by a name, expression and other options. Most commonly used slot options are:

- **:accessor** function-name
- **:initform** expression
- **:initarg** symbol

For example, let us define a Box class, with three slots length, breadth, and height.

```
(defclass Box ()
   (length
   breadth
   height)
)
```

Providing Access and Read/Write Control to a Slot

Unless the slots have values that can be accessed, read or written to, classes are pretty useless.

You can specify **accessors** for each slot when you define a class. For example, take our Box class:

```
(defclass Box ()
   ((length :accessor length)
      (breadth :accessor breadth)
      (height :accessor height)
   )
)
```

You can also specify separate **accessor** names for reading and writing a slot.

```
(defclass Box ()
   ((length :reader get-length :writer set-length)
      (breadth :reader get-breadth :writer set-breadth)
      (height :reader get-height :writer set-height)
   )
)
```

Creating Instance of a Class

The generic function **make-instance** creates and returns a new instance of a class.

It has the following syntax:

```
(make-instance class {initarg value}*)
```

Example

Let us create a Box class, with three slots, length, breadth and height. We will use three slot accessors to set the values in these fields.

Create a new source code file named main.lisp and type the following code in it.

```
(defclass box ()
   ((length :accessor box-length)
      (breadth :accessor box-breadth)
      (height :accessor box-height)
   )
)
(setf item (make-instance 'box))
(setf (box-length item) 10)
(setf (box-breadth item) 10)
(setf (box-height item) 5)
(format t "Length of the Box is ~d~%" (box-length item))
(format t "Breadth of the Box is ~d~%" (box-breadth item))
(format t "Height of the Box is ~d~%" (box-height item))
```

When you execute the code, it returns the following result:

```
Length of the Box is 10
Breadth of the Box is 10
Height of the Box is 5
```

Defining a Class Method

The **defmethod** macro allows you to define a method inside the class. The following example extends our Box class to include a method named volume.

Create a new source code file named main.lisp and type the following code in it.

```
(defclass box ()
   ((length :accessor box-length)
      (breadth :accessor box-breadth)
      (height :accessor box-height)
      (volume :reader volume)
   )
)

; method calculating volume

(defmethod volume ((object box))
   (* (box-length object) (box-breadth object)(box-height object))
)

 ;setting the values

(setf item (make-instance 'box))
(setf (box-length item) 10)
(setf (box-breadth item) 10)
(setf (box-height item) 5)

; displaying values

(format t "Length of the Box is ~d~%" (box-length item))
(format t "Breadth of the Box is ~d~%" (box-breadth item))
(format t "Height of the Box is ~d~%" (box-height item))
(format t "Volume of the Box is ~d~%" (volume item))
```

When you execute the code, it returns the following result:

```
Length of the Box is 10
Breadth of the Box is 10
Height of the Box is 5
Volume of the Box is 500
```

Inheritance

LISP allows you to define an object in terms of another object. This is called **inheritance.** You can create a derived class by adding features that are new or different. The derived class inherits the functionalities of the parent class.

The following example explains this:

Example

Create a new source code file named main.lisp and type the following code in it.

```
(defclass box ()
   ((length :accessor box-length)
      (breadth :accessor box-breadth)
      (height :accessor box-height)
      (volume :reader volume)
   )
)
; method calculating volume
(defmethod volume ((object box))
   (* (box-length object) (box-breadth object)(box-height object))
)

;wooden-box class inherits the box class
(defclass wooden-box (box)
((price :accessor box-price)))

 ;setting the values
(setf item (make-instance 'wooden-box))
(setf (box-length item) 10)
(setf (box-breadth item) 10)
(setf (box-height item) 5)
(setf (box-price item) 1000)

; displaying values

(format t "Length of the Wooden Box is ~d~%" (box-length item))
(format t "Breadth of the Wooden Box is ~d~%" (box-breadth item))
(format t "Height of the Wooden Box is ~d~%" (box-height item))
(format t "Volume of the Wooden Box is ~d~%" (volume item))
(format t "Price of the Wooden Box is ~d~%" (box-price item))
```

When you execute the code, it returns the following result:

```
Length of the Wooden Box is 10
Breadth of the Wooden Box is 10
Height of the Wooden Box is 5
Volume of the Wooden Box is 500
Price of the Wooden Box is 1000
```