Filed on: {Indigo}<Loops>LoopsCourse>LoopsCourseSummary.bravo
Last edited: GB June 14, 1983 6:47 PM

LOOPS Course Summary

Accessing Objects and Variables

($ *name*)	evaluates to the object or class named *name*.
($! *atom*)	evaluates to the object or class whose name is the value of *atom*
(@ *accessExpr*)	evaluates to the value of the instance variable, class variable, or property of these referred to by *accessExpr* in self.
(@ *obj accessExpr*)	evaluates to the value of the instance variable, class variable, or property of these referred to by *accessExpr* in *obj*
(_@ *accessExpr newValue*)	sets the value of the variable accessed by *accessExpr* in self to *newValue*
(_@ *obj accessExpr newValue*)	sets the value of the variable accessed by *accessExpr* in *obj* to *newValue*

accessExpr is the concatenation of any combination of the following with evaluation strictly left to right

:*ivName*	instance variable *ivName*
::*cvName*	class variable *cvName*
:,*propName*	value of property *propName*
.*selector*	value returned by sending the unary message *selector*

N.B. a ! (bang) after any of the puctuation in the four lines above will cause the atom following it to be evaluated and that value to be used as the name. Within an *accessExpr* a lisp variable is prefixed with a backslash "\"
(i.e. **::fee.fie:!\foe:,fum** will get the value of CV **fee** of self and send it the message **fie**, then it will get the instance variable whose name is the value of the lisp variable **foe** from the object returned by the message fie, then it will get the value of property **fum** of that IV)

Defining and Editing Classes

(DC *className supersList*)	(_ ($ Class) New *className supersList*)	create a class with name *className* and supers *supersList*
(EC *className*)	(_ ($ *className*) Edit)	edit the class definition of class *className*

Defining and Editing Methods

(DM *className selector*)	creates a function with the name *className.selector* to be used by the method called by *selector* and puts you in the editor
(DM *className selector fnName*)	causes the function with the name *fnName* to be used by the method called by *selector*
(EM *className selector*)	edit the method used by *selector* in class *className*

Creating, Editing, and Inspecting Instances

(_ *class* New)		creates a new instance of *class*
(_ *class* New *'name*)		creates a new instance of *class* with the name *name*
(_ *obj* Edit)	(EI *obj*)	edit *obj*
(_ *obj* Inspect)	(INSPECT *obj*)	create an inspect window for *obj*
(_New *class selector arg1 ... argN*)		create a new instance of *class* and sends it the the message *selector* with arguments *arg1 ... argN*

Sending Messages

(_ *obj selector arg1 ... argN*)	send *obj* the message *selector* with arguments *arg1 ... argN*
(_Super *obj selector arg1 ... argN*)	in method *selector* invokes super method for that *selector* with arguments *arg1 ... argN*
(_SuperFringe *obj selector arg1 ... argN*)	invokes all the immediate super methods of *obj* for that *selector* with the arguments *arg1 ... argN*
(_! *obj expr arg1 ... argN*)	send *obj* the message whose selector is the value of *expr* with the arguments *arg1 ... argN*

Active Values

#(*localState getFn putFn*)	*localState* is where the value is stored (this may be another active value) *getFn* is the function called on read access and *putFn* is called on write access the value returned by *getFn* in the value of the get operation and *putFn* has responsibility for changing the value of *localState* using the function PutLocalState

Debugging

(BreakIt *obj varName*) break whenever the instance variable *varName* of *obj* is accessed

(UnBreakIt *obj varName*) remove the break on variable *varName* of *obj*

(BreakMethod *className selector*) break whenever the method *selector* is used by any instance of class *className*

(TraceMethod *className selector*) trace whenever the method *selector* is used by any instance of class *className*

(UNBREAK *onlyMostRecentFlg*) standard Lisp function to unbreak or untrace methods

(BreakIt *obj varName propName type breakOnGetAlsoFlg*) break whenever the variable *varName* of *obj* is accessed

(TraceIt *obj varName propName type breakOnGetAlsoFlg*) trace whenever the variable *varName* of *obj* is accessed

(UnBreakIt *obj varName propName type*) remove the break on variable *varName* of *obj*

To attach a gauge and monitor a variable:

(_New ($ *gaugeType*) Attach *obj ivName selector*) attaches a gauge of type *gaugeType* to the instance variable *ivName* of *obj*

Rules

^F gets you into the Rule Executive

(OK gets you out of it and UE puts you in the User Executive (where OK will get you back again))

Variables are accesed by using the access expressions as defined above

accessExpr	gets value of variable (do not use @)
accessExpr_newValue	variable accessed gets *newValue*
\lispVarName	for referring to lisp variables use backslash
.selector	sends unary message to self (unary message is one that requires no arguments besides self)

(DefRSM *className selector*) creates a new rule set for the class *className* invoked by *selector* and places you in the rule editor

(_ *ruleSet* CopyRules *'newRuleSetName*) copies the ruleset *ruleSet* into a new one called *newRuleSetName*

(_ *ruleSet* ER) ER(*ruleSet*) edit *ruleSet*

(ListRuleSets *className*) generates a listing of all the rule sets defined for the class *className*

Browsers

(Browse *classList*) creates a browser window for the class lattice structure of the classes in *classList* and their descendants

left or middle button in title area of the browser window updates the lattice structure

Left Mouse Button gets pop-up menu to print information about class structure and methods

Middle Mouse Button gets pop-up menu to aid in generating new classes or methods

An asterisk at the end of the name of any item in the menu signifies that there are multiple options for this item

To use the default option, click the left button, for a menu of options click the middle button (i.e. EM* will get a menu with EM and EM!)

To copy from class to class use the left button to "BoxNode" of recepient class then with the middle button menu select the "Move" item with the middle button to get a menu for either copying of moving of IVs, CVs, Methods, or RuleSets

"Specialize" on the middle button menu will create a new subclass of the one selected and ask for a name in the prompt window

"DefineMethod" on the middle button menu will create a new method for that class and prompt for its selector

Saving and Restoring Files

(FILES?) Lisp will ask you to assign a *filename* to each entity it does not already have a file name for Type yes to specify the file names. For each entity type the *filename* to save it or] to not have it saved

LineFeed (LF) means the same as the previous entity

(MAKEFILE *filename*) saves the file on the file server on the directory currently connected

(LOAD *filename*) loads the file from the file server on the directory currently connected