■ Confusion about doc strings in DEFMETHOD.

People often do

```
(defmethod Area ((Rect Rectangle))
  "WIDTH times HEIGHT of the rectangle"
  (* (Width Rect) (Height Rect)))
```

without clearly thinking about what that might mean. Some people think it will make a doc string on the *generic* function that **(documentation 'Area 'function)** or the equivalent emacs keystrokes will retrieve. Others vaguely expect it to make a doc string on each separate method, and that the emacs doc-string retrieving keystroke (which just calls **documentation**) will somehow automagically be able to figure out which method it applies to. In fact, Lisp will accept this, but put the documentation on the method *object*, which beginners probably know nothing about. Use the **:documentation** entry in **defgeneric** to put a doc string on the generic function.

■ Invalid :initargs are accepted by MAKE-INSTANCE.

On SPARC machines, both Lucid Common Lisp 4.1.1 (both the production and development compilers) and Harlequin LispWorks 3.1.0 accept unknown **:initarg**'s without complaint, even at the highest **safety** settings.

```
(defclass Foo ()
 ((Slot-1 :accessor Slot-1 :initarg :Slot-1
:initform 5)))

; typo: SLOT1 instead of SLOT-1
(setq Test (make-instance 'Foo :Slot1 10))
[No error message]

(Slot-1 Test) ==> 5
```

This is a bug in the implementation; all implementations are supposed to flag this as an error.

■ Forgetting the class has to exist before any method that specializes upon it.

Lisp programmers are used to being able to define functions in any order, where even if **Foo** calls **Bar**, **Foo** can be defined first. But

```
(defmethod Area ((Rect Rectangle)) ...)
(defclass Rectangle (Polygon) ...)
```

is illegal. You have to define the class first.

■ Changing a method to apply to a more general class does not supersede previous method.

E.g. a user writes

```
(defmethod Half-Area ((Rect Filled-Rectangle))
 (/ (Area Rect) 2))
```

Then they notice that this functionality could apply to all Rectangles, not just Filled-Rectangles. So they change the class, with their intuition being that they are *replacing* the old definition, when in fact they are *adding* a new, less-specific method. They then later add a call to **float** to avoid getting a ratio back.

```
(defmethod Half-Area ((Rect Rectangle))
  (float (/ (Area Rect) 2)))
```

Then they are puzzled as to why their new definition appears not to have taken effect, since they are testing it on an instance of Filled-Rectangle, which still gets the old, more-specific definition.

Common Blunders

■ Omitting parens in arglist in DEFMETHOD.

Writing

```
(defmethod Area (Sq Square) ...)
```

instead of

```
(defmethod Area ((Sq Square)) ...)
```

Lisp will accept the former, and think that you have two unspecialized arguments instead of one argument specialized as a **`Square`**.

■ Missing parens around slot definition list in DEFCLASS.

Writing

```
(defclass Rectangle (Polygon)
  (Width ...)
  (Height ...))
```

instead of

```
(defclass Rectangle (Polygon)
  ((Width ...)
   (Height ...)))
```

Lisp will not accept the former, but the error message is not necessarily clear.

■ Forgetting empty slot definition list if you don't define local slots in DEFCLASS.

Writing

```
(defclass Square (Rectangle))
```

instead of

```
(defclass Square (Rectangle) ())
```

Lisp will not accept the former.

■ Referring to class name instead of instance variable in DEFMETHOD.

Writing

```
(defmethod Area ((Sq Square))
  (* (Width Square) (Width Square)))
```

instead of

```
(defmethod Area ((Sq Square))
  (* (Width Sq) (Width Sq)))
```

Lisp may give a warning about an unknown free variable, but probably won't even do that if you type the **`defmethod`** directly into the Lisp Listener (Lucid doesn't). So you might not get an error until run-time.

■ Forgetting accessors are functions and thus could conflict with built-in function names.

E.g. writing

```
(defclass Graphical-Object ()
  ((Position :accessor Position)))
```

Lisp will not accept this since you cannot redefine the built-in **`position`** function.

■ Putting the new value last instead of first in the definition of a SETF method.

Writing

```
(defmethod (setf Area) ((Sq Square)
                        (New-Area number))
  (setf (Width Sq) (sqrt New-Area)))
```

instead of

```
(defmethod (setf Area) ((New-Area number)
                        (Sq Square))
  (setf (Width Sq) (sqrt New-Area)))
```

Lisp will accept the former, and then users are puzzled as to why **`(setf (Area Square-1) 10)`** doesn't work.

■ Putting the new value last instead of first in a call to a :writer method.

E.g given

```
(defclass Circle ()
  ((Radius :reader Radius
           :writer Set-Radius
           :initform 5)))
(setq Circle-1 (make-instance 'Circle))
```

Writing

```
(Set-Radius Circle-1 10)
```

Instead of

```
(Set-Radius 10 Circle-1)
```