Throws a DeprecatedError. Use this to indicate that the enclosing method has been replaced by a better one (possibly in another class), and that it will likely be removed in the future. Unlike other errors, DeprecatedError only halts execution if `Error.debug == true` . In all cases it posts a warning indicating that the method is deprecated and what is the recommended alternative.

Discussion:

```
foo {
    this.deprecated(thisMethod, ThisOrSomeOtherObject.findMethod(\foo);
    ... // execution of this method will continue unless Error.debug == true
}

// For a class method:
*bar {
    this.deprecated(thisMethod, OtherClass.class.findMethod(\bar));
    ...
}
```

Printing and Introspection

.post

Print a string representation of the receiver to the post window.

```
"hello".post; "hello".post; "";
```

.postln

Print a string representation of the receiver followed by a newline.

```
"hello".postln; "hello".postln; "";
```

.postc

Print a string representation of the receiver preceded by comments.

```
"hello".postc; "hello".postc; "";
```

.postcln

Print a string representation of the receiver preceded by comments, followed by a newline.

```
"hello".postcln; "hello".postcln; "";
```

.postcs

Print the compile string representation of the receiver, followed by a newline.

```
"hello".postcs; "hello".postcs; "";
```

.dump

Print a detailed low level representation of the receiver to the post window. Except for the List class, this method is not overridden in other classes. Any of the MetaClasses, Classes and Instances can be used by this method.

```
Meta_Object.dump // the meta class of the class Object
Object.dump      // the class called Object
Object.new.dump  // an istance of the class Object
```

Discussion:

- The detailed low level format and information varies depending on the receiver.
- Some instance objects, especially unique objects, return the class name and value (also low data if necessary) of the dumped object:

Float

```
1.0.dump
```

64-bit version of SuperCollider returns:

```
   Float 1.000000   00000000 3FF00000
-> 1.0
```

The last two groups of an 8-digit integer are the raw hexadecimal representation of the 64-bit double value according to IEEE 754 Floating Point (https://ieeexplore.ieee.org/document/8766229). Each part is represented as follows:

```
                        raw hexadecimal representation
                          of the 64-bit double value
                        ______________________________

Float   -1.000000       00000000              3FF00000
|       |               |                     |
class   decmial         significant part      exponent part
        representation  (mantissa)            with sign bit
```

Integer

```
1.dump
```

```
   Integer 1
-> 1
```

Char

```
$1.dump
```

```
   Character 49 '1'
-> 1
```

The integer between *Character* and *'1'* is the ASCII value of that character.

Symbol

```
\1.dump
```

```
   Symbol '1'
-> 1
```

- Some instance objects return more detailed information, such as
  - addrress in virtual memory (the hexadecimal number prefixed with 0x),
  - *garage collector color* (gc),
  - *data format type* (fmt),
  - *flags for immutablity, finalization and garbage collector debug sanity check* (flg),
  - *size class* (set),
  - and so on (the information on the second and subsequent lines varies depending on the class to which the instance belongs)

  with the class name of the instance:

Array

```
[1, 2].dump;
```

```
Instance of Array {    (0x1552c9558, gc=78, fmt=01, flg=00, set=02)
  indexed slots [2]
      0 : Integer 1
      1 : Integer 2
}
-> [1, 2]
```

List

```
List[1, 2].dump;
```

```
List's array:
Instance of Array {    (0x13b3cb5b8, gc=6C, fmt=01, flg=00, set=02)
  indexed slots [2]
      0 : Integer 1
      1 : Integer 2
}
-> List[1, 2]
```

Set

```
Set[1, 2].dump;
```

```
Instance of Set {    (0x1489e2068, gc=A4, fmt=00, flg=00, set=02)
  instance variables [2]
    array : instance of Array (0x13b458838, size=4, set=2)
    size : Integer 2
}
-> Set[2, 1]
```