```
// 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:

```
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'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]
```

System Information

.gcInfo

Posts garbage collector information in a table format.

Discussion:

- flips: the number of times the GC "flipped", i.e. when it finished incremental scanning of all reachable objects
- collects: the number of partial collections performed
- nalloc: total number of allocations
- alloc: total allocation in bytes
- grey: the number of "grey" objects, i.e. objects that point to reachable objects and are not determined to be (un)reachable yet

Then for each size class: numer of black, white and free objects, total number of objects and the total set size.

```
flips 241  collects 689096   nalloc 40173511   alloc 322496998   grey 346541
0  bwf t sz:    882      0 368573   369455    2955640
1  bwf t sz:   6197    122 5702377   5708696   91339136
2  bwf t sz:    947      4 1500009   1500960   48030720
3  bwf t sz:   8056  65201 301800   375057   24003648
4  bwf t sz:   4047    145   3457     7649     979072
5  bwf t sz:    422      1    431      854     218624
6  bwf t sz:    124      2     72      198     101376
7  bwf t sz: 153504      1      0   153505   157189120
8  bwf t sz:     22      0      0       22      45056
9  bwf t sz:      5      0      0        5      20480
10  bwf t sz:      5      0      0        5      40960
12  bwf t sz:      2      0      0        2      65536
13  bwf t sz:      1      0      0        1      65536
19  bwf t sz:      1      0      3        4   16777216
tot bwf t sz: 174215  65476 7876722   8116413   341832120
```

You can also query the amount of free memory with `Object.totalFree` and dump the currently grey objects with `Object.dumpGrey` . More memory status methods are: largestFreeBlock, gcDumpSet, and gcSanity.

Iteration

.do(function)