Null Object Pattern

In Null Object pattern, a null object replaces check of NULL object instance. Instead of putting if check for a null value, Null Object reflects a do-nothing relationship. Such Null object can also be used to provide default behaviour in case data is not available.

The concept of null objects comes from the idea that some methods return null instead of real objects and may lead to having many checks for null in your code.

In Java and C++, the key to the Null Object pattern is an abstract class that defines the interface for all objects of this type. The Null Object is implemented as a subclass of this abstract class. Because it conforms to the abstract class' interface, it can be used any place this type of object is needed. Null object should not have any state.

In Eolang, the concept of null does not exist as every object is meant to dataize to a value, and as such given a value on creation. As classes and interfaces do not exist here either, the closest implementation in Eolang would be to have every object implement a null attribute that either dataizes to a Boolean or a string representing a lack of value/data or whatever the default value of an object may be. In this case, there may still be checks to see if null is true or false or contains the expected string.

EO Implementation

```
+package sandbox
+alias stdout org.eolang.io.stdout

[] > null
  "null" > @

[args...] > appNull
  stdout > @
    if.
      args.isEmpty
      null
      args.get 0
```

output:

run.cmd

null

Decorator design pattern

Decorator pattern allows a user to add new functionality to an existing object without altering its structure. This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.

In Eolang, a copy of an object can be made, and new functionality be added. Here, the original object represents the decorator.

Eolang Implementation

```
1.  +package sandbox
2.  +alias stdout org.eolang.io.stdout
3.
4.  [] > carsDecorate
5.    8 > @
6.
7.  [num] > someCars
8.     decorateWithMoreCars num > @
9.
10.    [number] > decorateWithMoreCars
11.      add. > @
12.        carsDecorate
13.        number
14.
15. [args...] > appDecorator
16.   stdout > @
17.     someCars (args.get 0)
```

In this example, the object “someCars” increases the number of cars in decarDecorator is for itself.

output:

run.cmd 5

13

It can be concluded that decorator design pattern naturally exits in Eolang.