DieHandle new vs. self
class new

When classes are first class citizen
S. Ducasse, L. Fabresse, G. Polito, and P. Tesone

http://www.pharo.org
Goal

- `self` represents the receiver
- Classes receive messages too
To support

(DieHandle new add: (Die faces: 4); yourself)
+ (DieHandle new add: (Die faces: 6); yourself)

We defined + as

DieHandle >> + aDieHandle
| handle |
handle := DieHandle new.
self dice do: [ :each | handle addDie: each ].
aDieHandle dice do: [ :each | handle addDie: each ].
^ handle
What happens when subclassing?

DieHandle << #MemoDieHandle
...

(MemoDieHandle new add: (Die faces: 4); yourself)
  + (MemoDieHandle new add: (Die faces: 6); yourself)
> aDieHandle

- We get a DieHandle instance back and not a MemoDieHandle instance!
- Current DieHandle»+ always returns an instance of DieHandle (hardcoded class use) even if the receiver is a subclass
Solution 1: Creating a hook method

DieHandle >> + aDieHandle
| handle |
handle := self handleClass new.
self dice do: [:each | handle addDie: each ].
aDieHandle dice do: [:each | handle addDie: each ].
^ handle

DieHandle >> handleClass
^ DieHandle

A subclass may redefine handleClass

MemoDieHandle >> handleClass
^ MemoDieHandle
Solution 1: Creating a hook method

(MemoDieHandle new add: (Die faces: 4); yourself) + (MemoDieHandle new add: (Die faces: 6); yourself) > aMemoDieHandle

We get an instance of the subclass!
But we can do better!

Pros:
- Extensibility

Cons:
- In each subclass we should redefine the hook method `handleClass`
- This is tedious and error prone (developer might forget)
Solution 2

```smalltalk
DieHandle >> + aDieHandle
| handle |
handle := self handleClass new.
sel self dice do: [ :each | handle addDie: each ].
aDieHandle dice do: [ :each | handle addDie: each ].
^ handle

DieHandle >> handleClass
^ self class
```

- `self class` always returns the class of the receiver (it works for subclasses too!)
- We get instances of the same kind of the receiver
Summary

- Do not hardcode class use

- Encapsulate class use in a self send (a hook)
- Extensible solution but requires redefinition

- Return the class of the receiver
- Gracefully adapt to future subclasses
- Still extensible by redefinition

---

```smalltalk
DieHandle >> + aDieHandle
| handle |
handle := DieHandle new.
...

... handle := self handleClass new.
...

DieHandle >> handleClass
^ DieHandle

DieHandle >> handleClass
^ self class
```