2do recuperatorio, 1er cuatrimestre de 2015

Se extendió la clase `OrderedCollection` con una subclase `AutoMapList`, la cual, al recibir un mensaje, primero intenta responderlo como lo haría una `OrderedCollection`, y si no lo entiende, lo delega a sus elementos, siempre y cuando todos puedan responderlo. De otra forma, falla como fallaría habitualmente. En otras palabras, es una lista que puede 'mapear' mensajes a sus elementos de manera implícita.

- Definir el método `AutoMapList >> respondsTo: aSelector` que devuelva verdadero si la lista **o** todos sus elementos responden al mensaje con selector `aSelector`.
- Definir lo necesario para que los mensajes se redirijan cuando corresponda. Por ejemplo:
  - `(AutoMapList with: 1 with: 4) size` devuelve 2.
  - `(AutoMapList with: 1 with: 4) + 10` devuelve una AutoMapList que contiene al 11 y al 14.
  - `(AutoMapList with: 1 with: 4) lala` produce una excepción.

Solución

```
AutoMapList>>respondsTo: aSelector
      ^super respondsTo: aSelector or: [self allRespondTo: aSelector]

AutoMapList>>doesNotUnderstand: aMessage
      ^(self allRespondTo: aMessage selector)
            ifTrue: [ self collect: [ :each | aMessage sendTo: each ] ]
            ifFalse: [ super doesNotUnderstand: aMessage ]

AutoMapList>>allRespondTo: aSelector
      ^self allSatisfy: [ :each | each respondsTo: aSelector ]
```

Último ejercicio: clases unitarias

En este ejercicio incorporaremos al lenguaje Smalltalk la capacidad de crear clases unitarias (en inglés *singleton*). Así como cada clase puede crear subclases de sí misma al recibir el mensaje `subclass:instancaVariableNames:classVariableNames:package:`, ahora también podrá crear clases con una única instancia.

Implementar el método de instancia `singletonSubclass: nombreSubclase` para la clase `Class`.

El comportamiento esperado es el siguiente: cuando una clase reciba el mensaje `singletonSubclass:` con un símbolo, debe crear una subclase de sí misma cuyo nombre sea el símbolo pasado como parámetro. Además, la nueva clase deberá redefinir el método `new`, de manera que si ya existe una instancia no cree una nueva, y en cambio devuelva la instancia ya existente.

Solución

```
Class>>singletonSubclass: aClassName
    | subclass |
    subclass := self subclass: aClassName
                    instanceVariableNames: ''
                    classVariableNames: 'instance'
                    package: self package name.
    subclass class compile: 'new
        instance ifNil:[instance := super new].
        ^instance'.
    ^subclass
```