Learning Object-Oriented Programming, Design and TDD with Pharo

Stéphane Ducasse

March 11, 2019

Layout and typography based on the sbabook LaTeX class by Damien Pollet.

Contents

Illustrations

CHAPTER 1

Fun with Capchas (not used for now)

Capchas are ways to control that a service is not accessed by a robot. In this chapter we propose to implement some capchas as a way to practice and learn Pharo. Several capcha will be implemented but first let us start with just manipulating strings.

CHAPTER 2

Reversing a string

The problem that we propose to you is to reserve a string. It can be useful to build capcha as we will show later and it is a small and simple one that we will teach how to interact with strings and with collections of objects.

```
'frog'
->
'grof'
```

2.1 Solution #4

There are several solutions to this problem and we will look at some of them but first let's look at some basic elements about strings.

2.2 Basic on strings

A string is a sequence of characters and the first one has the index 1 (and not zero like in some other languages)

```
'frog' size
--> 4
```

To access an element we use the message `at: anIndex`.

```
'frog' at: 1
-> $f
```

To set the value of a string element we use the message `at:anIndex put: aCharacter`

```
| s |
s := 'frog'.
s at: 1 put: $z.
s
-> zrog
```

2.3 Reversing

Now to reverse the string 'frog' we see that the `$f` index 1 should be put in the position 4, the `$r` index 2 in the position 3, the `$o` index 3 in the position 2 and `$g` index 4 should be put in the position 1. We should find a relation between the actual index of the character and its future index once the string will be reversed.

Let us think again about it by now using the original string size (i.e., 4). It seems that the target index is `size + 1 - source index`. Let us verify this hypothesis: `$f` whose index is 1, will be put in the index `4 + 1 - 1 = 4`, the `$r` whose index is 2, will be put in the position `4 + 1 - 2 = 3`, the `$o` whose index is 3, will be put in the position `4 + 1 - 3 = 2` and `$g` index 4 should be put in the position 1.

To solve this problem we also need to know how to perform a loop. The following script prints all the number from 1 to 10.

```
1 to: 10 do: [:i | Transcript show: i printString; cr].
```

The following script prints each element of the string 'frog' one after the other.

```
1
  to: 'frog' size
  do: [ :i | Transcript show: ('frog' at: i) ; cr ]
```

```
| sourceString targetString |
sourceString := 'frog'.
targetString := String new: sourceString size.

1 to: targetString size do: [ :i |
  targetString at: (sourceString size + 1) - i put: (sourceString
    at: i)
  ].
targetString
```

Now we can define a new method on the class `String`. In such case the sourceString variable is not necessarily anymore since this is the string receiving the message that will be the sourceString.

```
String>>captchaReversed

  | targetString |
  targetString := String new: self size.

  1 to: targetString size do: [ :i |
    targetString at: (self size + 1) - i put: (self at: i)
    ].
  ^ targetString
```

```
'youpi' captchaReversed
-> 'ipuoy'
```

Now `self size + 1` is invariant during the loop. It does not change so we can extract it outside the loop.

```
String>>captchaReversed

  | targetString n |
  targetString := String new: self size.
  n := (self size + 1).
  1 to: targetString size do: [ :i |
    targetString at: n - i put: (self at: i)
    ].
  ^ targetString
  ]]]

Now looking at how it is implemented in Pharo we see the following
    definition defined in the superclass of ==String==.


[[[
SequenceableCollection>>reversed
  "Answer a copy of the receiver with element order reversed."
  "Example: 'frog' reversed"

  | n result src |
  n := self size.
  result := self species new: n.
  src := n + 1.
  1 to: n do: [:i | result at: i put: (self at: (src := src - 1))].
  ^ result
```

`self species new: n.` makes the code working on several different collection. `species` returns a class of the same species than the receiver of the message, the class `Array` if the receiver is an array, the String class if the receiver a string. So result points on a new string or collection.

CHAPTER 3

Palindroms

The following example brings an interesting idea for another string manipulation for captcha: asking the user to enter an palindrome. Palindrome are words or sentences which are symmetrical. For example, 'civic', 'refer' and 'No lemon, no melon' are palidromes.

```
'civic' captchaReversed
->
'civic'
```

Here is a way to check if a string is a capcha.

```
captchaIsAnagram
  "Returns true whether the receiver is an anagram.
  'anna' captchaIsAnagram
    true
  'andna' captchaIsAnagram
    true
  'avdna' captchaIsAnagram
    false
  "
  1
    to: self size//2
    do: [ :i | (self at: i) = (self at: self size + 1 - i)
            ifFalse: [ ^false ]
        ].
  ^true
```

Now again the expression `self size + 1` is constant.

```
captchaIsAnagram
  "Returns true whether the receiver is an anagram.
  'anna' captchaIsAnagram
    true
  'andna' captchaIsAnagram
    true
  'avdna' captchaIsAnagram
    false
  "
  | n |
  n := self size + 1.
  1
    to: self size//2
    do: [ :i |
      ( self at:  i ) = ( self at:  n - i)
        ifFalse: [ ^false ]
        ].
  ^true
```

Bibliography