Parcial CAP

Curs 2021-22 (9/11/2021) Duració: 1.5 hores

1.- (1 punt) Defineix *reificació*, *introspecció* i *intercessió*. Dóna exemples d'introspecció i intercessió en Smalltalk.

Solució:

Les definicions són a les transparències.
- Introspecció: Poder *veure* la pila d'execució amb **`thisContext`**.
- Intercessió: Poder *canviar* la pila d'execució, gràcies a **`thisContext`**.

2.- (2 punts) Fes petits bocinets de Smalltalk (per ser executats al *Playground*) per respondre les següents qüestions:

- Per a cada classe de la imatge, escriure al **`Transcript`** el nom de la classe i la quantitat de mètodes que té (en un format com, p.ex. **`'OrderedCollection -- 61'`**).
- Obtenir una col·lecció amb tots els mètodes que fan servir en el seu codi **`#callcc:`**
- Trobar quantes classes implementen el mètode de selector **`#name`**
- Trobar quantes classes tenen al menys un mètode sobreescrit (*overriden*) en alguna subclasse.

Nota: *Recordeu que totes les classes hereten els getters per a les variables d'instància de la classe* **`Behavior`***, superclasse de totes les metaclasses. També us pot interessar mirar si la classe* **`CompiledMethod`** *té algún mètode que us pugui ser útil.*

Solució:

```
a) SystemNavigation default allClassesDo:
      [ :c | (c name , ' -- ' , c methodDict size asString) traceCr ]

b) SystemNavigation default allSendersOf: #callcc:.

c) (SystemNavigation default allClassesImplementing: #name) size

d) (SystemNavigation default allClasses select: [ :c |
       ((c methodDict) select:
             [ :m | m isOverridden ]) size > 0 ]) size.
```

3.- (2 punts) Executeu aquest codi al *Playground*:

```
| globalVariable bloc b |
globalVariable := 'Primera assignació'.
bloc := [ :s |  [ (globalVariable , ' ', s) traceCr ] ].
b := bloc value: '1'.
b value.
globalVariable := 'Segona assignació'.
b value.
b := bloc value: '2'.
b value.
```

Què surt? Explica amb precisió, però també breument, per què observeu el que observeu.

Solució:

Surt:
```
'Primera assignació 1'
'Segona assignació 1'
'Segona assignació 2'
```

bloc és un *closure* que captura **globalVariable**. Penseu que capturar una variable no significa capturar també el seu valor, així que si la variable capturada canvia de valor, això es veurà reflectit en l'execució de **bloc**. A més, **bloc** retorna un bloc sense paràmetres que captura el paràmetre **s** de **bloc**. Aquest bloc retornat es quedarà amb el valor donat a **s** fins que no tornem a avaluar **bloc** amb un altre paràmetre. La raó és la mateixa, la variable capturada pot canviar de valor.

4.- (2 punts) Executeu aquest codi al *Playground*:

```
| b cont |
b := Continuation callcc: [ :cc | cont := cc. false ].
#(1 2 3 4 5) do: [ :each | (b ifFalse: [ (5-each) asString ]
                            ifTrue:  [ each asString ]) traceCr ].
b ifFalse: [ cont value: true ].
```

Digueu què us surt al **Transcript**. Per què surt el que heu vist que surt?

Solució:

Apareix al **Transcript** el següent:
```
'4'
'3'
'2'
'1'
'0'
'1'
'2'
'3'
'4'
'5'
```
I la raó és que quan s'avalua el bloc argument del **#callcc:** retorna **false** (a part de guardar la continuació, és a dir, el fet d'assignar un valor a **b**). Així, la primera passada pel **#do:** escriurà 5 menys el nombre (**5-each**) al **Transcript**. Però en ser **b** igual a **false** executarem l'avaluació de la continuació, que assigna **true** a **b**. I continua amb **b** valent **true**.

5.- (3 punts) Afegiu un mètode a la classe **#BlockClosure** anomenat **#whileWithBreak:** que funcioni de manera similar al mètode **#whileTrue:** que ja coneixem: El receptor del missatge és un bloc sense paràmetres que s'ha d'avaluar a un booleà, i l'argument és un bloc sense paràmetres que representa el cos del bucle, que es repetirà mentre el receptor sigui **true**.

La diferència és que a **#whileWithBreak:** dins el cos del bucle (l'argument) podem utilitzar la següent expressió:

(#breakWhile binding) value: <expressió> ⇨ Atura el bucle **whileWithBreak:** i retorna el valor resultant d'avaluar **<expressió>**

Fixeu-vos que, al contrari que **#whileTrue:**, de **#whileWithBreak:** s'espera que retorni un resultat.

També, fixeu-vos que necessitareu utilitzar variables dinàmiques amb la classe **Binding** que ja vam veure a classe.

Mireu aquest exemple, similar a l'exemple que vam veure de **BlockWithExit** (classe que **no** podeu fer servir en aquesta resposta), excepte que **#whileWithBreak:** aquí retorna **true** si troba un nombre menor que 100:

```
| index found coll |
coll := Array new: 1000.
1 to: 1000 do: [ :i | coll at: i put: 1000 atRandom ].
index := 1.
found := [ index < 1000 ] whileWithBreak: [
                                    | each |
                                    each := coll at: index.
                                    each traceCr.
                                    (each < 100) ifTrue: [ (#breakWhile binding) value: true ].
                                    index := index + 1 ].
```

Aquest exemple escriu al **Transcript** els nombres triats a l'atzar que hi ha a **coll** (que aquí és un **Array**) fins que troba un nombre menor que 100, moment en que atura el bucle retornant **true**, que serà el nou valor de la variable **found**.

Solució:

A la classe **BlockClosure** afegim:

```
whileWithBreak: aBlock
     ^ Continuation callcc: [ :cc |
          #breakWhile bindTo: cc in: [ self whileTrue: aBlock ] ]
```

També podem fer-ho sense continuacions:

```
whileWithBreak: aBlock
     #breakWhile bindTo: [ :x | ^ x ] in: [ self whileTrue: aBlock ]
```