Parcial CAP

Curs 2018-19 (15/XI/2018) **Duració: 2 hores**

1.- (1 punt) Defineix *introspecció*, *intercessió*, digues què significa *reificar* i explica per quina raó no podem fer intercessió de la pila d'execució en Java.

2.- (1.5 punts) Si seleccionem aquest programa al Workspace i fem `Ctrl-p`...

```
| collection |
collection := OrderedCollection new.
#(1 2 3) do: [ :index |
      | temp |
      temp := index.
      collection add: [ temp ] ].
collection collect: [ :each | each value ]
```

el resultat és: `an OrderedCollection(1 2 3)`. En canvi, si seleccionem aquest programa al Workspace i fem `Ctrl-p`...

```
| collection temp |
collection := OrderedCollection new.
#(1 2 3) do: [ :index |
      temp := index.
      collection add: [ temp ] ].
collection collect: [ :each | each value ]
```

el resultat és: `an OrderedCollection(3 3 3).`

Explica i justifica la diferència en el resultat.

3.- (1.5 punts) A la plana 318 del capítol 14 del llibre *Deep into Pharo*, anomenat *Blocks: a Detailed Analysis*, quan fa referència a l'ús del retorn (recordeu, `^ expressió`) dins d'un bloc (és a dir, quelcom similar a `[... ^ expressió ]`), diu:

The evaluation of the block returns to the ***block home context sender*** *(i.e., the context that invoked the method creating the block)*

I teniu un exemple bastant aclaridor del que això significa. A la plana 320 teniu explicats els riscos d'utilitzar retorns dins de blocs.

Vull que escriviu codi que il·lustri el cas en que l'ús del retorn dins d'un bloc surt malament, és a dir, que genera un error (i que no sigui, literalment, l'exemple que hi ha en el llibre).

4.- (1.5 punts) Explica i justifica la relació entre aquesta invocació a `#callcc:`

```
     Continuation callcc: [ :k | expressió ]
```

i aquesta invocació a `#callcc:`

```
     Continuation callcc: [ :k | k value: expressió ]
```

Per simplificar suposarem que **no** s'invoca `k` dins d'***expressió***

5.- (1.5 punts) Escriviu un programa en Java que, passant el nom d'una classe com a paràmetre de la línia de comandes, digui si aquesta classe és abstracta.

6.- (3 punts) Definirem dues versions del `#whileTrue:` amb continuacions que ja vam veure a classe:

a)
```
BlockClosure >> whileTrueCCa: aBlock
     | cont tmp |
                              "Aquí està l'única diferència"
     cont := Continuation callcc: [ :cc | cc ].
     self value ifTrue: [ aBlock value.
                    Transcript show: ('inside whileTrueCC: ', tmp asString);cr.
                    tmp := tmp + 1.
                    cont value: cont]
                ifFalse: [^ nil].
```

b)
```
BlockClosure >> whileTrueCCb: aBlock
     | cont tmp |
                              "Aquí està l'única diferència"
     cont := Continuation callcc: [ :cc | cc ].
     self value ifTrue: [ aBlock value.
                    Transcript show: ('inside whileTrueCC: ', tmp asString);cr.
                    tmp := tmp + 1.
                    cont value: cont]
                ifFalse: [^ nil].
```

Si ara avaluem al Workspace:

```
| n |
n := 4.
[ n > 0 ] whileTrueCCa: [ n := n-1 ]
```

el resultat és:

```
inside whileTrueCC: 0
inside whileTrueCC: 0
inside whileTrueCC: 0
inside whileTrueCC: 0
```

Si ara faig el mateix al Workspace, però amb `BlockClosure >> #whileTrueCCb:` :

```
| n |
n := 4.
[ n > 0 ] whileTrueCCb: [ n := n-1 ]
```

el resultat és:

```
inside whileTrueCC: 0
inside whileTrueCC: 1
inside whileTrueCC: 2
inside whileTrueCC: 3
```

Mireu d'entendre què passa i especuleu sobre les possibles raons d'aquest comportament. És una pregunta oberta, ja que la resposta no s'ha explicat a classe, però amb el que sabeu haurieu de poder deduir-ne una resposta aproximada.