Pràctica CAP Q1 curs 2016-17
Corutines

- A realitzar en grups de **2 persones**.
- A entregar com a molt tard el **25 gener de 2017**.

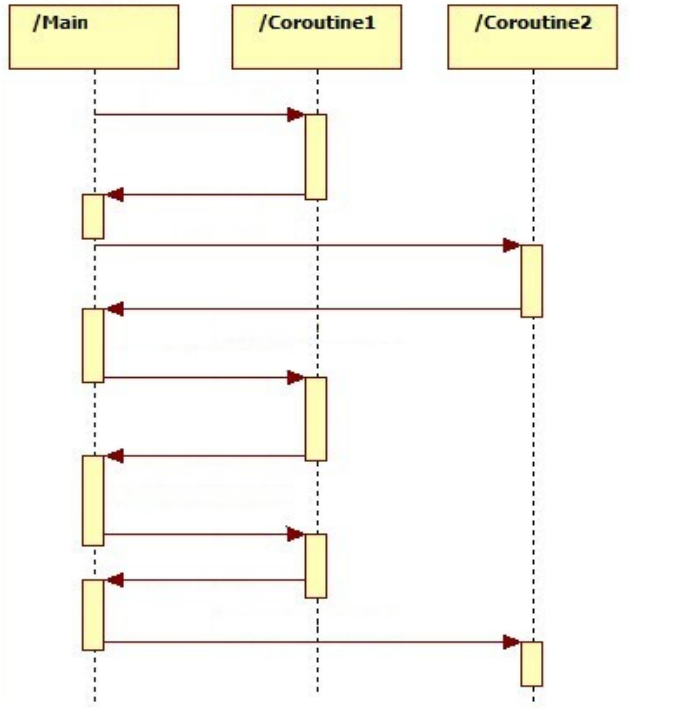
Descripció resumida:
tl;dr ⇒ Aquesta pràctica va de continuacions.
La pràctica de CAP d'enguany serà una investigació del concepte general d'estructura de control, aprofitant les capacitats d'introspecció i intercessió que ens dóna Smalltalk. Estudiarem les conseqüències de poder guardar la pila d'execució (a la que tenim accés gràcies a la pseudo-variable thisContext) per fer-la servir i/o manipular-la. El fet de poder guardar i restaurar la pila d'execució d'un programa ens permet implementar qualsevol estructura de control i implementar la versió més flexible i general de les construccions que manipulen el flux de control d'un programa: les continuacions. Utilitzant les continuacions implementarem una estructura de control anomenada Corutina (*coroutine*).

Material a entregar:
tl;dr ⇒ Amb l'entrega del codi que resol el problema que us poso a la pràctica NO n'hi ha prou. Cal entregar un informe i els tests que hagueu fet.
Haureu d'implementar el que us demano i entregar-me finalment un **informe** on m'explicareu, què heu aprés, i *com* ho heu arribat a aprendre (és a dir, m'interessa especialment el codi lligat a les proves que heu fet per saber si la vostra pràctica és correcte). Les vostres respostes seran la demostració de que heu entés el que espero que entengueu. El format de l'informe és lliure, i el codi que m'heu d'entregar me'l podeu entregar de diverses maneres: via un fitxer .st obtingut d'un File Out, via un paquet .mcz o via un paquet a SmalltalkHub. Qualsevol d'aquestes maneres és vàlida. Utilitzareu Pharo 3.0 (pharo.org) per fer la pràctica.

Nota: Abans de començar, llegiu i estudieu amb atenció el capítol 14, *Blocks: A Detailed Analysis*, del llibre *Deep into Pharo* (deepintopharo.com). També repasseu el que vam explicar a classe de continuacions. Per a la part b) de la pràctica us anirà bé l'article *Stable Marriages by Coroutines*, de Lloyd Allison (la referència és *Information Processing Letters 16 (1983), 61-65*, però us l'adjunto).

Enunciat:

La idea de l'estructura de control anomenada corutina és la següent: imaginem una funció (o procediment, o subrutina) tal i com ja les coneixem de C o C++. Les funcions s'invoquen, executen el seu cos, i quan acaben retornen. Si les tornem a invocar, torna a executar-se tot el cos de la funció, i quan acaba retorna. Les corutines funcionen diferent: Quan una corutina C_1 invoca una altra corutina C_2, s'atura i espera que se la torni a invocar. Si això passa, l'execució de C_1 es reprén just en el moment en que va invocar C_2; si ara C_1 torna a invocar C_2, l'execució de C_2 es reprén en el moment que va decidir invocar a un altre corutina. La idea es pot veure en aquest gràfic:



Hi ha diverses maneres de definir les corutines, en funció de diverses característiques i la literatura sobre el tema és nombrosa. Nosaltres ens limitarem a una definició senzilla, adaptada a Smalltalk i a la OOP, que ens permetrà jugar amb el concepte[1]. Això és el que us demano a l'enunciat, tot seguit.

Cal fer una classe: **`Coroutine`**, tal que construeixi objectes que es comportin com a corutines (en aquest sentit, podriem pensar que són *com una mena de* blocs). Per a això tenim un mètode **`Coroutine class >> #maker:`** que farem servir per instanciar aquesta classe. Caldrà passar un bloc com a paràmetre, que és on posarem el codi de la corutina. Aquest bloc serà un bloc de dos paràmetres:

```
<instancia de corutina> ← Coroutine maker: [ :resume :value | . . . ]
```

En els punts suspensius és on posarem el codi de la corutina.

1 Es fa servir en el capítol 17 del llibre *Scheme and The Art Of Programming* **https://www.cs.unm.edu/~williams/cs357/springer-friedman.pdf**

Aquesta corutina, via el paràmetre **`:resume`**, pot invocar altres corutines:

```
resume value: <nom corutina> value: <valor passat a la corutina invocada>
```

ja que **`resume`** ha de ser *un bloc amb dos paràmetres*: el primer és una referència a una altra corutina, el segon és el valor que se li passarà a aquesta corutina com a valor de retorn de la crida a corutina que va fer el darrer cop que es va executar.

La manera com s'invoca una corutina serà mitjançant un mètode de la classe **`Coroutine >> #value:`**. En un programa fet amb corutines només cal arrencar la primera corutina, a partir d'aquest moment les corutines es comencen a cridar entre elles. El bloc que passarem com a paràmetre **`:resume`** al bloc amb que es construeix la corutina (el paràmetre de **`#maker:`**) *si el pensem bé*, pot ser sempre el mateix i independent de qualsevol codi que posem dins les corutines. També cal que penseu que la corutina, quan s'invoca per primer cop, ha de posar en marxa el codi especificat en el bloc que es passa com a paràmetre a **`#maker:`**, però després, quan s'invoca retornant d'una crida anterior (amb **`resume value: c value: v`**), ha de fer una altra cosa (ha de fer **`c value: v`**, *però no només això!* aquesta és una de les coses que heu de pensar).

Veiem un exemple. Si executem:

```
| a b c |
a := Coroutine maker: [ :resume :value |
      Transcript show: 'This is A'; cr.
      Transcript show: 'Came from ', (resume value: b value: 'A'); cr.
      Transcript show: 'Back in A'; cr.
      Transcript show: 'Came from ', (resume value: c value: 'A'); cr.  ].

b := Coroutine maker: [ :resume :value |
      Transcript show: '       This is B'; cr.
      Transcript show: '       Came from ', (resume value: c value: 'B'); cr.
      Transcript show: '       Back in B'; cr.
      Transcript show: '       Came from ', (resume value: a value: 'B'); cr.  ].

c := Coroutine maker: [ :resume :value |
      Transcript show: '           This is C'; cr.
      Transcript show: '           Came from ', (resume value: a value:'C');cr.
      Transcript show: '           Back in C'; cr.
      Transcript show: '           Came from ', (resume value: b value: 'C');cr.].

a value: nil. "en aquest exemple, el valor que li passem a a és irrellevant"
```

El resultat al **`Transcript`** és:

```
This is A
       This is B
           This is C
Came from C
Back in A
           Came from A
           Back in C
       Came from C
       Back in B
Came from B
```

Així doncs, el que us demano és:

a) [**7 punts**] Implementeu la classe **`Coroutine`** i feu que l'exemple de l'enunciat funcioni correctament (si a més vosaltres penseu altres exemples, millor).

b) [**3 punts**] Implementeu una solució al problema dels matrimonis estables (*stable marriage problem*) utilitzant corutines:

Donat un conjunt de N homes, N dones i una llista de preferències per a cada home i cada dóna, trobeu un aparellament estable pel grup. Un aparellament és inestable si hi ha un home i una dóna que no estan aparellats però que es prefereixen l'un a l'altre (i l'altre a l'un) més que a la seva actual parella.

Us passo un article que us explica el problema i us proposa una solució *amb corutines*. Només cal que adapteu la solució de l'article a Smalltalk, utilitzant les corutines implementades a l'apartat anterior.