Parcial CAP

Curs 2020-21 (30/11/2020) Duració: 2 hores

1.- (1 punt) Defineix *introspecció* i *intercessió*. Dóna exemples d'introspecció i intercessió en Smalltalk i Java (un de cada a cada llenguatge).

Les definicions són a les transparències.
- Introspecció:
  Smalltalk: Poder *veure* la pila d'execució amb **`thisContext`**.
  Java: Poder saber quins mètodes i atributs té una classe a Java (amb **`getDeclaredMethods`** i **`getDeclaredFields`** de **`Class`**).
- Intercessió:
  Smalltalk: Poder *canviar* la pila d'execució, gràcies a **`thisContext`**.
  Java: Poder interceptar les invocacions a mètode amb l'ús de la classe **`java.lang.reflect.Proxy`**.

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

- Obtenir una col·lecció amb tots els mètodes (instàncies de **`RGMethodDefinition`**) que es corresponen (és a dir, que tenen com a nom) al selector **`#inject:into:`**
- Trobar quants mètodes hi ha en el sistema que NO estan implementats a la màquina virtual (que NO són *primitives*)
- Obtenir una col·lecció amb totes les classes que implementen **`#inject:into:`**
- Trobar quantes classes hi ha en el sistema.

```
a) SystemNavigation default allImplementorsOf: #inject:into:
b) (SystemNavigation default allMethods size)
            - (SystemNavigation default allPrimitiveMethods size)
   N'hi ha 92119 en una imatge de Pharo 6.1 nova
c) SystemNavigation default allClassesImplementing: #inject:into:
d) (SystemNavigation default allClasses) size N'hi ha 6481 en una
imatge de Pharo 6.1 nova
```

3.- (2 punts) Un bloc en Smalltalk té variables *locals* (paràmetres, variables declarades dins el bloc), i variables *lliures*, que pertanyen al context en el que s'ha creat el bloc, però no són declarades *dins* el bloc. Sabem que els blocs en Smalltalk són realment el que hem anomenat *closures*. Aleshores:

a) Què significa que un bloc d'Smalltalk sigui en realitat una *closure*?

b) La pseudo-variable **`self`** es pot fer servir dins un bloc, si aquest es crea dins un mètode. Aquesta pseudo-variable és capturada per un bloc/*closure* que la faci servir? Dona un exemple que justifiqui la teva resposta.

a) Significa que els blocs *capturen* el seu context lèxic, és a dir, les variables lliures del bloc han de poder-se trobar en el context on es crea el bloc, i aquest les captura podent sobreviure a la desaparició del context que ha creat el bloc.

b) La pseudo-variable `self` **també** és capturada per un *closure*.Teniu un exemple a les planes 312 i 313 del capítol *Blocks: A Detailed Analysis*.

4.- (2.5 punts) Executeu aquest codi al Playground:

```
| b cont |
b := Continuation callcc: [ :cc | cont := cc. false ].
#(1 2 3 4 5) do: [ :each | (b ifFalse: [ 'NO' ]
                              ifTrue: [ each asString ]) traceCr ].
b ifFalse: [ cont value: true ].
```

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

Apareix al **`Transcript`** el següent:

```
'NO'
'NO'
'NO'
'NO'
'NO'
'1'
'2'
'3'
'4'
'5'
```

I la raó és que quan s'avalua el bloc paràmetre 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 bucle escriurà **`'NO'`** 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.- (2.5 punts) Escriviu un mètode que, passant dos *class object* com a paràmetres, diguem-ne **`C1`** i **`C2`**, retorni **`true`** si la classe **`C1`** té algun mètode *propi* que necessiti algun paràmetre de classe **`C2`**.

```
public boolean pregunta_5 (Class c1, Class c2) {
 if (c1 != null) {
     try {

       Method m[] = c1.getDeclaredMethods();

       for (int i = 0; i < m.length; i++) {
         Class pt[] = m[i].getParameterTypes();

         for (int j = 0; j < pt.length; j++) {
           if (pt[j].equals(c2)) return true;
         }
```

```
        }

      }
      catch (Throwable e) {
       System.err.println(e);
      }
 }
 return false;
}
```