Parcial CAP

Curs 2015-16 (9/XII/2015) **Duració: 2 hores**

1.- (*2 punts*) Implementa la classe **`BinaryTree`** en Smalltalk, amb les operacions **`left`** (fill esquerra), **`right`** (fill dret) i **`preorder`** (recorregut en preordre).

2.- (*2 punts*) Escriviu un fragment de codi (per ser executat en el *workspace*) que ens digui quants mètodes tenen la **`String`** '**`this`**' en el seu codi font.

3.- (*2 punts*) A classe vam veure un mètode anomenat **`#haltIf: aSymbol`** que permetia fer un **`halt`** només quan a la cadena de crides hi havia un mètode amb **`aSymbol`** com a nom. Feu ara un mètode **`#haltIfTrue: aBlock`** on **`aBlock`** és un bloc *sense paràmetres*. Aleshores, **`self haltIfTrue: [...]`** s'aturarà només si el resultat d'avaluar **`[...]`** és **`true`**. A quina classe ha d'anar aquest mètode?

4.- (*2 punts*) Aquest codi podria ser un test de continuacions:

```
test
 | i cont |
 i := 0.
 i := i + (Continuation callcc: [:cc | cont := cc. 1]).
 self assert: i ~= 3.
 i = 2 ifFalse: [ cont value: 2 ]
```

El resultat és que el test és *verd*. Qué fa? Quina propietat de les continuacions s'està comprovant?

5.- (*2 punts*) Detalla les modificacions que hauries de fer a **`TracingIH.java`** per a que la traça només fos dels mètodes d'una classe concreta, diguem-ne **`A`**, i que a més aquests mètodes tinguéssin un nom amb un nombre senar de caràcters.

Solucions:

1.-
```
Object subclass: #BinaryTree
   instanceVariableNames: 'node left right'
   classVariableNames: ''
   category: 'Parcial'
```

'left' i 'right' són getters

```
preorder
 ^(OrderedCollection with: self node) ,
          self left preorder ,
          self right preorder
```

on cal afegir a `UndefinedObject >> preorder  ^ ''`

Constructors:

```
node: anObject left: aBinaryTree right: bBinaryTree
     ^ self new left: aBinaryTree; right: bBinaryTree; node: anObject

node: anObject
     ^ self node: anObject left: nil right: nil

node: anObject left: aBinaryTree
     ^ self node: anObject left: aBinaryTree right: nil
```

etc...

2.-
```
(SystemNavigation default allClasses)
     inject: 0
     into: [ :coll :c |
               coll + (c methodDict select: [ :v |
                    (v sourceCode findString: 'this') ~= 0 ] ) size ]
```

i el resultat és 2544

3.-
```
Object >> haltIf: aBlock
     aBlock value ifTrue: [ Halt halt ]
```

4.- El que fa és tornar al moment de l'increment d'`i` i demostrar que el context original de la captura de la continuació és el que toca, on `i` valia `0` (i no `1`, com podriem haver pensat degut a l'increment inicial). El fet que l'`assert` es verifiqui així ho demostra (si el context no hagués estat l'original, `i` hagués valgut 3).

5.- Canviarem el mètode `invoke` per mirar si `target` és com cal:

```
public Object invoke(Object proxy, Method method, Object[] args)
     throws Throwable {
 Object result = null;

 // Suposem que la classe A pertany a algun paquet
 String classname = target.getClass().getName();
 String unqname = classname.substring(classname.lastIndexOf('.')+1);
 boolean trace=(unqname.equals('A') && (method.getName().length % 2 == 1)

 try {
  if (trace)
       System.out.println( method.getName() + "(...) called" );
  result = method.invoke( target, args );
 } catch (InvocationTargetException e) {
  if (trace)
       System.out.println(method.getName()+" throws "+e.getCause());
  throw e.getCause();
 }
 if (trace)
      System.out.println( method.getName() + " returns" );
 return result;
}
```