

Parcial CAP

Curs 2014-15 (18/XI/2014)

Duració: 2 hores.

1.- (2 punts) Tenim les següents definicions de les classes A, B:

```
Object subclass: #A
  instanceVariableNames: 'a'
  classVariableNames: ''
  category: 'Exercicis'
A subclass: #B
  instanceVariableNames: ''
  classVariableNames: ''
  category: 'Exercicis'
```

i els següents mètodes:

```
A >> a      ^ a - 2
A >> a: unNombre      a := unNombre
B >> a
  | x |
  ^ super a <= 0 ifTrue: [1]
    ifFalse: [ x := B new.
                x a: super a.
                x a + 1 ]
```

Respondre les següents qüestions:

1. Avalueu l'expressió `B new a: 3`; a pas a pas
2. Quin és el resultat de l'expressió:
| b |
(1 to: 10) collect: [:i | b := B new.
 b a: i.
 b a]
3. Podeu simplificar el mètode `a` de la classe `B` de manera que no contingui referències a `super`?
4. Expliqueu matemàticament què calcula el mètode `a` de la classe `B`

2.- (1 punt) Els mètodes `#new` i `#new:` són mètodes d'instància de la classe `Behavior`, tot i que és habitual que se'ls redefineixi en altres classes. Malgrat són mètodes d'instància de `Behavior`, usualment es redefeixen com a mètodes de classe. Per exemple, utilitzem `Array new: 5` per crear instàncies de la classe `Array`, i el mètode `#new:` l'estem enviant a la classe `Array`. Les redefinicions, doncs, s'acostumen a fer en el *Class side*. Això aparentment viola la "regla" que diu que en l'herència els mètodes d'instància s'hereten en l'*instance side* i els mètodes de classe s'hereten en el *Class side*.

Expliqueu per quina raó no hi ha res d'incorrecte en el fet de redefinir `#new` i `#new:` en el *Class side*.

3.- (1 punt) Expliqueu com un bloc pot cridar-se a ell mateix (aconseguint així blocs sense nom, anònims, però recursius). *No podeu suposar que el bloc ha estat assignat a una variable.*

4.- (2 punts) Escriviu un fragment de codi (per ser executat en un *Workspace*) per trobar totes les classe que implementin el mètode `#hash` o el mètode `#=` , però no tots dos a la vegada.

5.- (2 punts) Escriviu un mètode `find: astring` que, *enviat a una classe*, retorni una col·lecció de selectors tal que els mètodes corresponents contenen *aString* dins del seu codi font.

Exemple: El resultat d'executar al *Workspace* `Object find: 'reflect'` hauria de ser una col·lecció amb els selectors de tots aquells mètodes d'*Object* tals que la string *'reflect'* apareix en el seu codi font. En aquest cas obtindriem:

```
#(#perform:withArguments:inSuperclass: #perform:with:
#perform:withArguments: #perform:with:with: #perform:
#perform:with:with:with:)
```

A quina classe cal posar aquest mètode per a que *qualsevol* classe o metaclasse del sistema sigui capaç d'executar-lo? I on el posem, a l'*instance side* o al *Class side*?

6.- (2 punts) Feu un mètode que, donada una classe (una instància de *Class*, un *class object*), escrigui a la sortida estàndard el nom de tots els seus ascendents (superclasse, superclasse de la superclasse, etc) amb totes les interfícies que implementen.

Exemple: Podriem fer servir aquest mètode dins d'un programa per obtenir una sortida similar a:

```
$ java Problema6 java.lang.reflect.Method
```

```
La classe java.lang.reflect.Method es filla de java.lang.reflect.AccessibleObject
i implementa els interfaces:
  java.lang.reflect.GenericDeclaration
  java.lang.reflect.Member
```

```
La classe java.lang.reflect.AccessibleObject es filla de java.lang.Object
i implementa els interfaces:
  java.lang.reflect.AnnotatedElement
```

```
La classe java.lang.Object no es filla de ningú
i NO implementa cap interface
```

No cal que la sortida del mètode sigui *idèntica* a aquesta, només cal que proporcioni la mateixa informació.

