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 redefineixen 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 interficies 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 ningu
 i NO implementa cap interface
```

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

1.-1. L'expressió `B new a: 3; a` el que fa és avaluar **a-2**, crear una nova instància de **B** a la que s'assigna aquest valor **a-2** i tornar a cridar **a** sobre aquest nou objecte. Aquesta invocació retorna 1, per tant el resultat final, valor de **x a + 1**, és 2.
2.- **#(1 1 2 2 3 3 4 4 5 5)**
3.- Podem canviar-los trivialment per **a-2**.
4.- Calcula el resultat de dividir per 2 i, si el resultat no és exacte, arrodonir a l'enter més gran. Això queda força clar amb la resposta a la pregunta 2.

2.- La resposta està en el mecanisme de les metaclasses. Una classe és sempre instància de la seva metaclasse, que és on es busquen els missatges enviats a la classe. Tota metaclasse és subclasse (indirecte) de **Behavior**, i hereta, per tant, els mètodes **#new** i **#new:**.

3.- Des de dins del bloc podem fer referència a **thisContext closure**, on **closure** és un atribut de **MethodContext**. Recordeu que gràcies a això Pharo no fa servir la classe **BlockContext**, ja que a l'atribut **closure** guardo el bloc en cas que el context actual correspongui al context d'execució d'un bloc (és **nil** en altre cas). Podem avaluar el bloc amb **thisContext closure value: ...** (suposant que el bloc té un parametre, tot i que això no treu generalitat a la resposta).

4.- Probablement hi ha millors solucions, però aquesta és suficient...

```
((SystemNavigation default allClasses
     select: [ :cl | cl includesSelector: #hash])
                  reject: [ :cl | cl includesSelector: #=]) union:
((SystemNavigation default allClasses
     select: [ :cl | cl includesSelector: #=])
                  reject: [ :cl | cl includesSelector: #hash])
```

5.- Caldria posar-ho a **Behavior**, com a mètode d'instància.

```
find: aString
     ^ (self methodDict select: [ :v |
                    (v sourceCode findString: aString) ~= 0 ] ) keys
```

6.- El mètode podria ser similar a:

```
void printClassAndInterfaceHierarchy(Class c) {
   if (c != null) {
       try {
            // Aconsegueixo la classe pare i les interfaces implementades
            Class   spc = c.getSuperclass();
            Class inc[] = c.getInterfaces();

            // Escric la corresponent informació
            System.out.println("La classe " + c.getName() +
                               ((spc != null) ? (" es filla de " + spc.getName())
                                              : " no es filla de ningu"));

            System.out.println((inc.length > 0) ? "i implementa els interfaces: "
                                                : " i NO implementa cap interface");
            for (int i = 0; i < inc.length; i++)
                System.out.println("  " + inc[i].getName());

            System.out.println();
            // Pujo un nivell a la jerarquia per fer el mateix...
            printClassAndInterfaceHierarchy(spc);
        }
        catch (Throwable e) {
            System.err.println(e);
        }
   }
}
```