Parcial CAP

Curs 2013-14 (12/XII/2013)
Duració: 2 hores.
Cada pregunta val 1 punt.

Smalltalk:

1.- Fes un mètode de la classe **`OrderedCollection`** amb un objecte qualsevol com a argument, que afegeixi l'objecte a la col·lecció només si aquest no hi és.

SOLUCIÓ:

```
addIfAbsent: anObject
    (self includes: anObject)
          ifFalse:[ self add: anObject ]
```

2.- Què resulta d'avaluar aquesta expressió? Justifica la teva resposta.

```
    ((1 to: 5) collect: [:i | [ |local| local := i ]])
        collect: [:each| each value]
```

SOLUCIÓ:

`(1 to: 5) collect: [:i | [ |local| local := i ]]` genera un array amb cinc closures **`[ |local| local := i]`**. Cada una d'elles ha estat creada com a resultat d'avaluar **`[:i | ... ]`** on el context lèxic inclou la variable **`i`**, paràmetre del bloc. Cada cop que hem creat un closure hem capturat una **`i`** *diferent*, amb un valor diferent (els elements de l'interval **`1 to: 5`**, i per tant en avaluar cada un d'aquests blocs el resultat ha estat diferent.

3.- Fes un mètode de la classe **`OrderedCollection`** amb una col·lecció de nombres naturals com a argument, que retorni una nova col·lecció amb els elements del receptor que es corresponguin a les posicions enumerades a la col·lecció que passem com a argument. Per exemple, si anomenem **`#atAll:`** al mètode que es demana,

```
        (OrderedCollection withAll: #(a b c d))  atAll: #(2 4)
```

tindrà com a resultat **`#(b d)`**

SOLUCIÓ:

```
atAll: aCollection
   | c |
   c := OrderedCollection new.
   aCollection do: [:i | c add: (self at: i)]
   ^c
```

respondre el mètode ja existent a Pharo **`SequenceableCollection>>atAll:`** es considerarà incorrecte ja que el que es demana aquí és més senzill (menys general).

Reflexió (general):

4.- Quines són les diferències entre intercessió i introspecció?

SOLUCIÓ: Està literalment explicat a les transparències que teniu.

Reflexió (Smalltalk):

5.- Com iteraries sobre tots els atributs (variables d'instància) d'un objecte? Quins mètodes faries servir? Pensa que l'objecte pot ser un objecte qualsevol.

SOLUCIÓ:
Utilitzant els mètodes **`#instSize`** i **`#instVarAt:`**, ja que no coneixem el nom de les variables d'instància. L'esquema seria quelcom de semblant a:

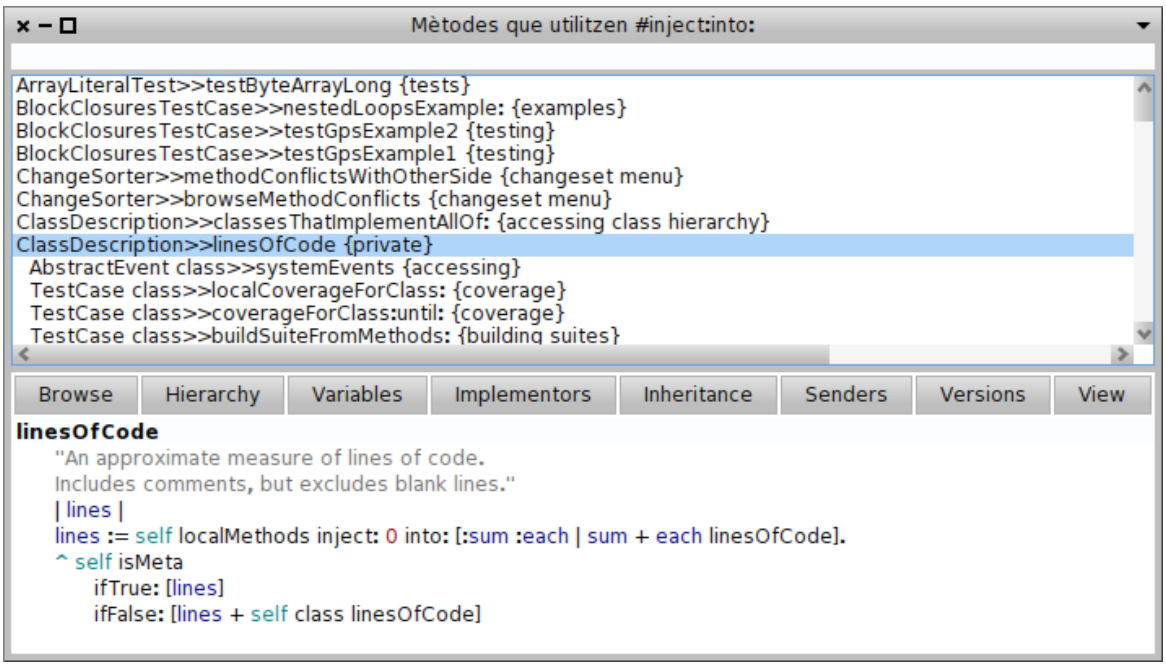
```
1 to: obj class instSize do: [:i | ... (obj instVarAt: i) ... ]
```

6.- Per quina raó s'avalua a **`true`** la següent expressió?

```
Class class class class == Class class class class class class
```

SOLUCIÓ:
Per la circularitat que hi ha entre **`Metaclass`** i la seva metaclasse **`Metaclass class`** en la relació "*ser-instància-de*".

7.- Fes un programa (per ser executat al *workspace*) que, donat un símbol que representarà el selector d'un mètode, generi un *browser* amb tots els mètodes que l'utilitzen dins el seu codi font. Per exemple, si el símbol fos **`#inject:into:`** el resultat hauria de ser:



SOLUCIÓ: A Pharo 1.4 (la versió de Pharo que fem servir a CAP) una solució seria:

```
| metode |
metode := #inject:into:.
((RBBrowserEnvironment new forClasses: (Object withAllSubclasses))
     selectMethods: [:m | m sendsSelector: metode])
     label: 'Mètodes que utilitzen #', metode ;
     open.
```

Reflexió (Java):

8.- Quin serà el resultat d'avaluar aquestes expressions? Justifica la teva resposta.

```
Class.class.isInstance(Class.class)
Class.class.isInstance(Object.class)
```

SOLUCIÓ:

En tots dos casos és **`true`**, degut a que **`java.lang.Class`** juga el paper de metaclasse, de la que tota classe és instància, fins i tot ella mateixa.

9.- Quina diferència hi ha entre els mètodes de la classe **`Class`**, **`getMethod`** i **`getDeclaredMethod`**? A què ens obliga si volem conèixer tots els mètodes d'una classe, propis i heretats, públics, privats, etc.?

SOLUCIÓ:

La diferència és que **getMethod** només proporciona mètodes *públics* declarats dins de la classe o heretats per la classe. D'altra banda, **getDeclaredMethod** permet obtenir qualsevol mètode declarat a la classe, i només aquests. Això ens obliga a fer servir **getDeclaredMethod** si volem obtenir absolutament tots els mètodes de la classe independentment de la seva visibilitat. A més, si també volem obtenir els mètodes heretats, hem de fer un recorregut recursiu cap a les superclasses per tal de obtenir-los tots.

10.- Feu un programa que, donat el nom d'una classe qualsevol, sigui capaç d'escriure tots els mètodes propis, amb els corresponents qualificadors, tipus de retorn, nom i paràmetres.

SOLUCIÓ:

```
Class c = Class.forName(<String: nom de la classe>);
Method m[] = c.getDeclaredMethods();
for (int i=0; i < m.length; ++i)
     System.out.println(m[i].toString());
```