FIB

CONCEPTES AVANÇATS DE PROGRAMACIÓ
Tema 3
Col·lecció de Problemes

Recull per Jordi Delgado
(Dept. CS, UPC)

Grau d'EI, 2022-23
FIB (UPC)

1.- Respon a les següents preguntes:

- Què és el lligam dinàmic?
- Una classe és un objecte?
- Quina és la diferència entre un missatge i un mètode?
- Hi ha alguna cosa a Smalltalk que NO sigui un objecte?
- Hi ha alguna cosa a Smalltalk que NO es faci per pas de missatges?
- Quina és l'arrel de la jerarquia de classes?
- Com s'indica que un mètode és "privat"?
- Què és una cascada? Quin és el seu valor?
- Com es defineixen les classes abstractes a Smalltalk?
- Quina és la diferència entre self i super?
- Com s'inicialitzen les variables d'instància en crear un objecte?

2.- Per a cada expressió, respon a aquestes qüestions:

Quin(s) és/són el(s) objecte(s) receptor(s)?
Quin(s) és/són el(s) selector(s) del(s) missatge(s)?
Quin(s) és/són el(s) argument(s)?
Quin és el missatge? o Quins són els missatges?
Quin és el resultat d'avaluar aquesta expressió?

- `5 + 6`
- `5 @ 6`
- `#($m $i $d $a) size`
- `Date today`
- `anArray at: 1 put: 'hola'`
- `anOrderedCollection inject: 1 into: [:acc :y | acc * y ]`
- `OrderedCollection new at: 1 put: 'un'; at: 2 put: 'dos'`
- `OrderedCollection new at: 1 put: 'un'; at: 2 put: 'dos'; yourself`
- `Transcript show: (2 + 1) printString.`
- `5@5 extent: 6.0 truncated @ 7`

3.- Quina mena d'objecte descriuen les expressions literals **`'Hola, món'`**, **`#HolaMón`** i **`#($H $o $l $a)`**?

4.- En les següents expressions, quins parèntesi són redundants?:

- `((5 + 6) + (2 * 5) + (2 - 1))`
- `(x isNil) ifTrue: [...] ifFalse: [...]`
- `(x = y) ifTrue: [...]`
- `(x includes: y) ifTrue: [...]`

5.- Si tenim el següent codi:

```
| unArray suma |
suma := 0.
unArray := #(21 23 53 66 87).
unArray do: [ :each | sum := sum + each ].
suma
```

Quin és el resultat final de **`suma`**? Com podriem reescriure el mateix però utilitzant l'indexat explícit de taules (amb el mètode **`#at:`**) per accedir als seus elements?

6.- Avalua les següents expressions:

- **(111111111111111111/1111111111111111112) + (1/1111111111111111112)**
- **2r1000**
- **16rFF**
- **256 printStringRadix:2**
- **(8 bitOr: 1) printStringRadix:2**
- **(8 bitAnd: 9)**
- **2e10 asInteger**
- **1 + 2; yourself**
- **1/12 + (1/6)**
- **1/12 + 1/6**
- **-2.5 truncated**

7.- Col·leccions. Feu mètodes que realitzin les següents accions:

- Compteu el nombre d'elements d'una taula (**Array**)
- Fer la mitjana d'una taula
- Construir una taula T a partir d'una altra A on $T_i = A_{2i-1} + A_{2i}$
- A partir d'una col·lecció C retornar la seva inversa
- Donats un bloc amb un paràmetre i una col·lecció C, cal retornar el resultat d'aplicar el bloc a tots els elements de la col·lecció
- Donat un element i una col·lecció, afegir l'element a la col·lecció només si aquest no hi és

8.- Obriu un **System Browser**:

- Busqueu la classe **Integer**
- Examineu la jerarquia de la classe **Integer**
- Busqueu el mètode **+** de la classe **Integer**
- Busqueu tots els mètodes **+** (totes les implementacions del mètode anomenat **+**)
- Busqueu tots els mètodes que utilitzen el sel·lector **+**

9.- Suposem que tenim la definició de tres classes anomenades **Un, Dos** i **Tres**:

```
Object subclass: #Un
     instanceVariableNames: ''
     classVariableNames: ''
     package: 'Exercicis'

Un subclass: #Dos
     instanceVariableNames: ''
     classVariableNames: ''
     package: 'Exercicis'

Dos subclass: #Tres
     instanceVariableNames: ''
     classVariableNames: ''
     package: 'Exercicis'
```

amb els següents mètodes definits:

```
Un >> a      ^1
Dos >> a     ^2
Dos >> b     ^super a
```

```
Tres >> a      ^3
Tres >> b      ^self a
Tres >> c      ^super b
Tres >> d      ^super a
```

Què s'escriurà al Transcript quan avalui el següent?:

```
un := Un new.
dos := Dos new.
tres := Tres new.
Transcript show: un a; cr;
          show: dos a; cr;
          show: dos b; cr;
          show: tres b; cr;
          show: tres c; cr;
          show: tres d; cr
```

10.- Quins són els resultats d'avaluar les següents expressions?

- **#(((1 2 3) 4) 5) first first first**
- **#(-1 2 -3 4) detect: [:x | x > 0]**
- **#(-1 2 -3 4) select: [:x | x > 0]**
- **#(1 ($a $b) 2) at: (#(1 ($a $b) 2) indexOf: #($a $b))**
- **#(1 2 3 (1 2 3) 2 3 (2 3) 3 (3)) copyWithout: 3**
- **#(1 2 3) raisedTo: 3**
- **#(1 2 3) * #(4 5 6)**
- **#(1 2 3) raisedTo: #(4 5 6)**

11.- Implementeu els següents mètodes:

- **#mig -** donada una **SequenceableCollection** retorna l'element del mig.
- **#swap:with:** - que té dos indexos **i** i **j** com a arguments i canvia l'element **i** pel **j** i el **j** per l' **i** a la col·lecció receptora.
- **#negated** - canvia el signe de tots els elements de la col·lecció
- **#copyWithFirst:** - que retorna una col·lecció idèntica a la receptora, però amb l'argument inserit en primer lloc
- **#inject:into:** - Ja sabeu com funciona, ara, sense mirar la implementació de Pharo, proveu d'implementar-lo vosaltres.
- **#anySatisfy:** - Retornarà **true** si existeix al menys un element dins la col·lecció tal que el bloc (d'un paràmetre) que passem com a argument retorna **true** quan l'apliquem a l'element
- **#atAll:** - Retornarà una col·lecció amb els elements corresponents a les posicions enumerades dins la col·lecció que passem com a argument. P.ex: **#(a b c d) atAll: #(2 4)** tindrà com a resultat **#(b d)**

12.- Suposem que en una *seqüència doble* hi ha dues seqüències formades pels simbols A, C, T, o G. Direm que la seqüència doble està *ben formada* si els simbols es corresponen en parelles A,T o G,C. És a dir, si a la posició **i** d'una seqüència hi ha una A, a la posició **i** de l'altra seqüència ha d'haver una T, si a la posició **i** d'una seqüència hi ha una G, a la posició **i** de l'altra seqüència ha d'haver una C, etc. Implementeu un mètode **#benFormada:** que

donada una seqüència formada pels simbols A, T, C, G retorni **`true`** si, donada una altra seqüència de A, T, C, G, les dues seqüències constitueixen una seqüència doble ben formada.

13.- Tenim les següents definicions de les classes **`A`**, **`B`** i **`C`**:

```
Object subclass: #A
     instanceVariableNames: 'a b'
     classVariableNames: ''
     package: 'Exercicis'
A subclass: #B
     instanceVariableNames: ''
     classVariableNames: ''
     package: 'Exercicis'
A subclass: #C
     instanceVariableNames: ''
     classVariableNames: ''
     package: 'Exercicis'
```

i els següents mètodes:

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

Per a les següents qüestions, mireu de trobar una solució *sense* escriure i executar el codi. Executeu-ho a l'ordinador només quan tingueu pensades les solucions.

- Avalueu l'expressió **`B new a: 3; a`** pas a pas
- Quin és el resultat de l'expressió:

```
| b |
(1 to: 10) collect: [ :i |
                      b := B new.
                      b a: i.
                      b a ]
```

- Podeu simplificar el mètode **`a`** de la classe **`B`** de manera que no contingui més que una referència a **`super`**?
- Podeu simplificar el mètode **`b`** de la classe **`C`** de manera que no contingui cap referència a **`super`**?
- Quin és el resultat de l'expressió:

```
| c |
(1 to: 10) collect: [ :i |
                        c := C new.
                        c b: i.
                        c b ]
```

- Expliqueu matemàticament què calculen els mètodes **a** de la classe **B** i el mètode **b** de la classe **C**
- Quin serà el resultat de la qüestió precedent si modifiquem el mètode **b** de la classe **C** de la següent manera:

```
C >> b
   | c |
   ^ super b = 0 ifTrue: [1]
                 ifFalse: [ c := C new.
                            c b: super b.
                            c b * super b + 1 ]
```

- Com caldria canviar el mètode **a** de la classe **B** per a que calculi la divisió entera per 3 del valor de la variable d'instància **a** (arrodonit cap a l'enter més gran si **a** no era múltiple de 3)?

14.- Implementeu en Smalltalk el TAD **ArbreBinariCerca** (em remeto als apunts d'EDA per a una explicació, descripció i implementació en **C++** del TAD).

15.- Implementeu en Smalltalk el TAD **CuaAmbPrioritat** (em remeto als apunts d'EDA per a una explicació, descripció i implementació en **C++** del TAD).

16.- Implementeu la classe **`Matriu`** (com a subclasse d'**`OrderedCollection`**). Ha de tenir els mètodes bàsics **`#new:`**, **`#at:`**, **`#at:put:`**. Podeu representar els parells de nombres amb la classe **`Point`**. Implementeu també el mètode **`#simetrica`**, que dirà si la matriu receptora és o no simètrica.

17.- Implementeu una subclasse d'**`Array`** tal que mantingui la història dels valors assignats a cada posició i permeti l'assignació de valors màxims i mínims a cada posició. Haureu d'implementar, sobre la classe:

```
Array variableSubclass: #ArrayAmbHistoria
     instanceVariableNames: ''
     classVariableNames: ''
     package: 'Exercicis'
```

els mètodes:

`#historia:` - retorna una col·lecció amb tots els valors que ha tingut la posició que es passa com a argument.

`#invertir` - Inverteix l'ordre dels valors de l'**`ArrayAmbHistoria`**

`#at:valorMax:` - assigna un valor màxim a una determinada posició (els elements que siguin assignats a aquesta posició no poden ser més grans que aquest valor màxim)

`#at:valorMin:` - assigna un valor mínim a una determinada posició

18.- Blocs: Què retornen aquests blocs avaluats tal com s'indica? (suposarem que cada bloc s'assigna a una variable anomenada **b**).

- Bloc: **`[ :x | | y | y := x. y > 0 ifFalse: [ y := -1*y ]. y ]`**

Avaluació: **`b value: -2`**
Avaluació: **`b value: 2`**

- Bloc: **`[ :x | | y | y := x. y > 0 ifFalse: [ y := -1*y ] ]`**
  Avaluació: **`b value: -2`**
  Avaluació: **`b value: 2`**
- Bloc: **`[ :x | ...<càlculs molt complicats>... . 1 ]`**
  Avaluació: **`b value: 10`**
  Avaluació: **`b value: 15 factorial`**

19.- Si seleccionem aquest programa al Playground i fem **`Ctrl-p`**...

```
| collection |
collection := OrderedCollection new.
#(1 2 3) do: [ :index |
     | temp |
     temp := index.
     collection add: [ temp ] ].
collection collect: [ :each | each value ]
```

el resultat és: **`an OrderedCollection(1 2 3)`**. En canvi, si seleccionem aquest programa al Playground i fem **`Ctrl-p`**...

```
| collection temp |
collection := OrderedCollection new.
#(1 2 3) do: [ :index |
     temp := index.
     collection add: [ temp ] ].
collection collect: [ :each | each value ]
```

el resultat és: **`an OrderedCollection(3 3 3).`**

Explica i justifica la diferència en el resultat.