Programación orientada a objetos. Segundo parcial 2001.

Viernes 29 de junio de 2001

1. Expliquen para qué son utilizadas las clases abstractas. ¿Tiene sentido definir un método abstracto que sea privado? Justifiquen su respuesta.

10 puntos

2. Sea una clase `ColoresPrimarios` con los métodos `amarillo`, `azul` y `rojo`, y una clase `ColoresCompuestos` con los métodos `naranja`, `violeta` y `verde`. Escriban en Smalltalk, el código necesario para tener una clase Colores que tenga métodos con un comportamiento exactamente igual a la clase en la que está definido.

```
Object subclass: #ColoresPrimarios
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''   !

!ColoresPrimarios class methods ! !

!ColoresPrimarios methods !

amarillo!
 ...

azul!
 ...

rojo! !
 ...

Object subclass: #ColoresCompuestos
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''    !

!ColoresCompuestos class methods ! !

!ColoresCompuestos methods !

naranja!
 ...

verde!
 ...

violeta! !
 ...
```

15 puntos

3. Existen lenguajes de programación orientada a objetos que permiten especificar cuando la resolución del método asociado a un selector se realiza empleando encadenamiento dinámico o estático. Algunos utilizan la palabra clave `virtual` para indicar que se realizará encadenamiento dinámico. ¿Cuándo es conveniente declarar los métodos con `virtual`? ¿Es correcto definir un método virtual y privado? Justifiquen su respuesta.

15 puntos

4. Consideren el siguiente método de clase `new`. Supongan que `initialize` es un método de instancia de esta clase.

```
new
  self new initialize;
    yourself.
```

   a. No funciona ¿Porqué?

   b. Escriban la versión corregida.

5 puntos

5. Imaginen que existe un lenguaje de programación orientada a objetos similar a Smalltalk llamado Xmalltalk en el que se declara el tipo de las variables así:

```
| a as String |
a := String new.
```

   Y en los argumentos en un método:

```
nombreMetodo: unString as String
...
```

   Consideren ahora que tienen las siguientes clases:

```
Object subclass: #Alfa
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: '' !

!Alfa class methods ! !

!Alfa methods !

isAlfa
  ^true! !

Alfa subclass: #Beta
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''   !

!Beta class methods ! !

!Beta methods !

isAlfa
  ^false!

isBeta
  ^true! !

Object subclass: #Gama
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: '' !

!Gama class methods ! !

!Gama methods !

gama: unArgumento as Alfa
  unArgumento isAlfa ifTrue: [^'alfa'] ifFalse: [^'beta']! !
```

Indiquen cuál sería el resultado obtenido evaluando cada bloque con “Show It”, suponiendo que el encadenamiento es dinámico. ¿Y si el encadenamiento es estático?

a.
```
| a as Gama b as Alfa |
a := Gama new.
b := Alfa new.
a gama: b.
```

b.
```
| a as Gama b as Beta |
a := Gama new.
b := Beta new.
a gama: b.
```

10 puntos

6. Consideren una clase `Perro` implementada en Smalltalk de la siguiente manera:

```
Object subclass: #Perro
  instanceVariableNames: 'cruzadoCon '
  classVariableNames: ''
  poolDictionaries: '' !

!Perro methods !

cruzadoCon
  "Retorna el perro con el cual fue cruzado este perro."
  ^cruzadoCon!
```

a. Implementen un método `cruzarCon:` que recibe como argumento una instancia de `Perro`. Agreguen en la clase todo lo que consideren necesario, incluyendo precondiciones y poscondiciones usando el método `error:`, de manera de asegurar:

- Que una perro se cruza con otro.
- Que los perros se cruzan con perros de la misma raza.
- Que los perros cruzados sean un casal.

b. Supongan ahora que los perros nacen conuna raza y mantienen esta raza durante toda su vida. ¿La clase perro tiene algún invariante? Si la respuesta es afirmativa, describan el invariante. ¿Es posible implementar los invariantes en Smalltalk? En caso afirmativo realicen los cambios que consideren necesarios en la clase Perro de la parte a.

20 puntos

7. Implementen los métodos necesarios para que un objeto de la clase `Persona` pueda responder lo siguiente:

- Su nombre en mayúsculas.
- Cantidad de vocales en su nombre.
- Cantidad de palabras en su nombre

La clase `Persona` está definida de la siguiente manera:

```
Object subclass: #Persona
  instanceVariableNames:
    'nombre apellido '
  classVariableNames: ''
  poolDictionaries: '' !

!Persona class methods ! !

!Persona methods ! !
```

15 puntos

8. En cada una de las siguientes características, indiquen al menos un lenguaje de programación orientada a objetos visto en clase, que tenga esa característica:
   a. Soporta interfaces como concepto separado de clases.
   b. Soporta encadenamiento estático y dinámico.
   c. Soporta herencia múltiple.
   d. Soporta la programación por contratos de forma nativa.
   e. Soporta garbage collection.

2 puntos por pregunta con lenguaje ajeno
1 punto por pregunta con lenguaje propio