Programación orientada a objetos. Examen julio 1999.

Miércoles 21 de julio de 1999.

1. En Smalltalk el estado de un objeto es privado e inaccesible a los demás objetos y el comportamiento es público y accesible a todos los objetos.

   a. ¿Cómo se hace en Smalltalk para convertir parte o todo el estado de un objeto de privado a público?

   b. Se dice que un método es "protegido" si es accesible a los objetos de la clase y a los de sus subclases pero no a los objetos de las demás clases. Smalltalk no soporta este concepto pero ¿cómo podría simularlo?

   Justifiquen sus respuestas.

2. El programador Armando Lío y la programadora Elba Gallo programaron sendas clases en Smalltalk como abstracción de una lapicera.

   La clase de Armando Lío tiene la siguiente definición:

```
Object subclass: #Lapicera
  instanceVariableNames: 'color marca precio'
  classVariableNames: ''
  poolDictionaries: ''
```

   La clase de Elba Gallo tiene la siguiente definición:

```
Object subclass: #Lapicera
  instanceVariableNames: 'color grosor escribiendo posicion'
  classVariableNames: ''
  poolDictionaries: ''
```

   Ambas clases tienen los métodos selectores y modificadores correspondientes a cada variable de instancia como es habitual.

   ¿Cuál de las dos es una buena abstracción? ¿Son las dos malas abstracciones? ¿Ambas pueden ser buenas abstracciones? Justifiquen sus respuestas.

3. Imaginen un lenguaje de programación orientado a objetos similar a Smalltalk -llamado Xmalltak- en el que se declara el tipo de las variables así:

```
| a: String b: Collection |
a := String new.
b := Collection with: a.
```

   Las asignaciones incorrectas son detectadas al evaluarlas con **Do it** o **Show it**. Xmalltalk considera que una asignación es incorrecta siguiendo el criterio de sustitución que se vio en clase.

   Al evaluar el siguiente código

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

   Xmalltalk inserta el texto "`assignment missmatch`" después del símbolo de asignación "`:=`" como sucede en Smalltalk con el texto "`invalid receiver`" o "`unfinished comment`":

```
|a: String |
a := assginment missmatch Integer new.
```

   Los mensajes con selector no implementado en ningún método del receptor son detectados al evaluarlos con Do it o Show it. Xmalltalk usa encadenamiento dinámico para la resolución de métodos en tiempo de ejecución.

   Al evaluar el siguiente código

```
|a: String |
...
a factorial
```

Xmalltalk inserta el texto "`invalid receiver`" antes del selector:

```
|a: String |
...
a invalid receiver factorial
```

Sean ahora dos clases en Xmalltalk definidas así:

```
Object subclass: #Alfa
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: '' !
! Alfa methods !
isAlfa
  ^true !

Alfa subclass: #Beta
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: '' !
! Beta methods !
isAlfa
  ^false !

isBeta
  ^true !
```

¿Qué ocurre al evaluar con **Show It** cada uno de los siguientes bloques de código en Xmalltalk?

a.-

```
| a: Alfa b: Beta |
a := Alfa new.
b := Beta new.
^a isAlfa and: [b isAlfa].
```

b.-

```
| a: Alfa b: Beta |
a := Alfa new.
b := Alfa new.
^a isAlfa and: [b isAlfa].
```

c.-

```
| a: Alfa b: Beta |
a := Beta new.
b := Beta new.
^a isAlfa and: [b isBeta].
```

d.-

```
| a: Alfa b: Alfa |
a := Alfa new.
b := Beta new.
^a isAlfa and: [b isAlfa].
```

e.-

```
| a: Alfa b: Beta |
a := Alfa new.
b := Beta new.
^b isAlfa and: [a isBeta].
```

Justifiquen sus respuestas.