PARADIGMAS DE PROGRAMACIÓN – EXAMEN FINAL – 15/02/2014

1. Se tiene la siguiente base de conocimiento:

```
tiene(juan, auto).
tiene(juan, moto).
tiene(pedro, moto).
tiene(moncho, bondi).
tiene(moncho, moto).
```

```
cuantosTiene(X, C):-
          findall( Y, tiene(X,Y), L),
          length(L,C).
envidia(A, B):- tiene(B,X),
      not(tiene(A,X)).
```

a. ¿Qué se obtiene como respuesta con las siguientes consultas? Justificar

```
?- tiene(juan, B).            ?- tiene(A, 3).
?- cuantosTiene(juan, B).     ?- cuantosTiene(A, B).
?- envidia(moncho, juan).     ?- envidia(A, B).
```

b. Se quiere saber si una persona tiene todo lo que tiene otra. Plantear una solución para que la siguiente consulta funcione y explicar qué conceptos se observan en la misma.

```
?- tieneTodoLoQueTiene(A, B).
A = juan,   B = pedro;
A = moncho, B = pedro.
```

2. Tenemos un sistema diseñado para ayudar a escapar a pequeñas naves espaciales, llamadas cápsulas, del planeta donde se encuentran[1]. Que una cápsula logre escapar a una galaxia consiste en que encuentre un lugar habitable en alguna de las estrellas de la mencionada galaxia y viaje hacia allí. Los lugares habitables son los planetas y asteroides que giran alrededor de una estrella y que cumplen con ciertas características.

```
#Capsula
>>escaparA: unaGalaxia
      unaGalaxia estrellas do: [:estrella |
            estrella sosHabitable ifTrue: [
                  self viajarA: estrella lugarHabitable.
                  ^ self. "esto hace que el método termine" ] ]
>>viajarA: unlugar
"Está implementado correctamente"
```

```
#Estrella
>>sosHabitable
      lugares do: [:unLugar | unLugar
sosHabitable ifTrue: [^true]].
      ^ false
>>lugarHabitable
      lugares do: [:unLugar | unLugar
sosHabitable ifTrue: [^unLugar]].
```

```
#Planeta
>>sosHabitable
      ^ hayLuna

#Asteroide
>>sosHabitable
      ^ diametro > 20000.
```

En un equipo de desarrollo en el que no sólo les preocupa que su sistema ande, sus integrantes conversan sobre las herramientas conceptuales que hacen a una buena solución. Elegir **una de tres afirmaciones** con la que estés de acuerdo y **justificar la respuesta**. Mejorar la solución en base a los problemas identificados. O sea, se debe: elegir, justificar y programar en cada caso.

[1] Kal-El y Kakaroto son potenciales usuarios de este sistema

a. **Encapsulamiento:**
   i. Estanislao: "En la solución hay cápsulas, entonces no se está rompiendo el encapsulamiento".
   ii. Froilán: "Se respeta el encapsulamiento porque los objetos cápsula tienen todo el comportamiento de la selección de lugares, que es su responsabilidad".
   iii. Florinda: "Se está rompiendo el encapsulamiento del objeto galaxia, ya que la cápsula es quien busca en las estrellas".
b. **Polimorfismo:**
   i. Estanislao: "Es necesario que planetas, asteroides y estrellas entiendan el mismo mensaje `sosHabitable`, para mantener el polimorfismo".
   ii. Froilán: "De acuerdo con que lo entiendan asteroides y planetas, pero en las estrellas el mensaje podría llamarse diferente, ya que ningún otro objeto le envía mensajes indistintamente a las estrellas y al resto".
   iii. Florinda: "Más aún, tampoco hace falta que sea igual entre planetas y asteroides. Podrían tranquilamente ser tres mensajes diferentes, ya que cada objeto sabe siempre de qué clase es".
c. **Errores**
   i. Estanislao: "Si la galaxia no tuviera estrellas habitables, deberíamos hacer que el sistema arroje un error".
   ii. Froilán: "No hace falta, ya da error sólo así como está".
   iii. Florinda: "¿Arrojar errores? ¿Para qué? Un buen sistema nunca da error".
d. **Declaratividad**
   i. Estanislao: “Podríamos reescribir los métodos de la estrella para mejorar la declaratividad”.
   ii. Froilán: “Me parece que no podemos hacer las cosas más declarativas ahí, ya que Smalltalk no tiene un motor, y la declaratividad está asociada a un motor”
   iii. Florinda: “Ojo, ya están suficientemente declarativos: entiendo perfectamente el algoritmo”.

3. Para el mismo problema que en el punto 2, se tiene una solución parcial hecha en Haskell:
```
escaparA capsula galaxia = (viajarA capsula . lugarHabitable . head . filter sosHabitable . estrellas) galaxia
```
   a. ¿Qué sucedería para cada una de las soluciones si la galaxia tuviera infinitas estrellas? Justifique conceptualmente en base al código dado (no valen justificaciones genéricas).
   b. Suponiendo que el código de `viajarA:` es el siguiente:
```
#CapsulaEscape>>viajarA: unLugar
      combustible := combustible - (unLugar distanciaA: miPosicion).
      miPosicion := unLugar posicion.
```
      i. Codificar la función `viajarA` que está utilizada en la solución en Haskell, suponiendo que una cápsula es una tupla (combustible,posición) y que existe la función `distanciaA`, que recibe una posición y un lugar.
      ii. ¿Qué concepto existe en el método `viajarA:` que no existe en la función `viajarA`? Explicar qué implica la presencia / ausencia del mismo.
   c. Esta solución (en Haskell) puede mejorarse en términos de abstracción y/o delegación. Identificar al menos una abstracción faltante (una o más nuevas funciones auxiliares) y reescribir la solución usándola/s.
   d. ¿Es posible reescribir la función anterior de la siguiente forma? Justificar conceptualmente, y relacionar al menos con el concepto de tipo.
```
escaparA capsula = viajarA capsula . lugarHabitable . head . filter sosHabitable . estrellas
```