Paradigmas de Programación - Examen Final 01/08/2015

Nota: Para aprobar es necesario obtener 8 puntos de 14 y al menos la mitad de los puntos en cada paradigma. En todas las respuestas sé puntual, sin perder el foco de lo que se pregunta. Las respuestas muy generales son tan malas como las incompletas.

EJERCICIO 1

Se tiene el siguiente código en Smalltalk para elegir algún buen alumno de un curso..

#Curso

Tenemos una versión en objetos.

```
buenAlumno
 |alumnosElegidos|
 alumnosElegidos := alumnos select: [:alumno | alumno nota = 10].
 ^ alumnosElegidos first
```

1. Explique lo que sucede al ejecutar las siguientes líneas de código suponiendo que miCurso no tiene alumnos con nota 10. *(1 punto)*
   ```
   ayudantes := Set new.
   ayudantes add: miCurso buenAlumno.
   ```
2. ¿Hay efecto/s colaterales en la solución? ¿Dónde? ¿Cómo afecta a otras partes del código? *(1 punto)*
3. Hay dos soluciones al mismo problema *(2 puntos)*:

En Prolog...
`buenAlumno(Alumno) :-`
`      nota(Alumno,10).`
	En Haskell...
	`buenAlumno alumnos = head (filter ((==10).nota) alumnos)`

En ambas nuevas soluciones:

a. ¿Qué sucede si hay muchos alumnos con nota 10? Justificar conceptualmente en ambos casos.
b. ¿Y si no hay ninguno? Justificar conceptualmente en ambos casos.

EJERCICIO 2

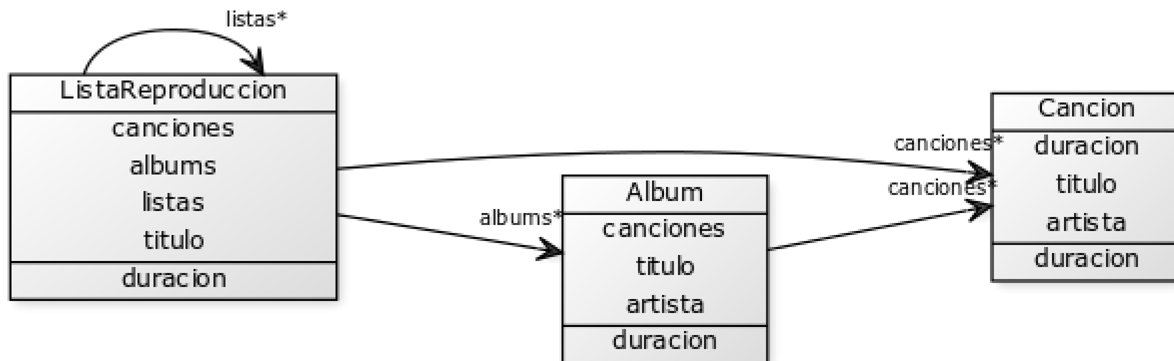
Se tiene la siguiente función en Haskell

```
f [ ] _ _ = [ ]
f (x:xs) a b
      | a (fst x) > b = snd x : f xs a b
      | otherwise = f xs a b
```

1. Mostrar dos ejemplos de invocación y respuesta de la función **`f`**, con diferentes tipos de datos y funciones *(1 punto)*
2. Desarrollar una definición alternativa de **`f`**, más **declarativa**. *(2 puntos)*
3. Justificar dónde se ve que la nueva versión es más declarativa. *(1 punto)*

EJERCICIO 3

Un sitio para escuchar música online permite a sus usuarios armar listas de reproducción con canciones, discos e incluso otras listas de reproducción propias o de otros usuarios. Se quiere saber cuál es la duración total de una lista de reproducción, para lo cual se tiene el siguiente código (también existen los getters para las variables):




```
#ListaReproduccion
>> duracion
  |duracionCanciones duracionAlbums duracionListas|
  duracionCanciones := canciones sum: [:cancion | cancion duracion ].
  duracionAlbums := albums sum: [:album | album duracion].
  duracionListas := listas sum: [:lista | lista duracion].
  ^ duracionCanciones + duracionAlbums + duracionListas
```

```
#Album
>> duracion
  ^ canciones sum: [:cancion | cancion duracion ].
```

1. El/la autor/a afirma que le puso a todos los métodos el mismo nombre para hacer polimórfica la solución. ¿Estás de acuerdo? En caso afirmativo, justificar. En caso contrario, modificar la solución para que sea polimórfica según tu propio criterio. *(1 punto)*

2. Se tiene una parte de una solución al mismo problema en Prolog.
```
duracion(album(_,_,Canciones),DuracionTotal):-
      findall(D,(member(C,Canciones),duracion(C,D)),Duraciones),
      sumlist(Duraciones,DuracionTotal).
```
   a. Completar la solución para cumplir con el requerimiento inicial de modo que se aproveche el polimorfismo. *(2 puntos)*
   b. Analizar la inversibilidad de los predicados definidos en la solución final. *(1 punto)*

3. En la solución de objetos, se agrega el requerimiento de poder agregar un elemento a la lista de reproducción de un usuario. Para los usuarios gratuitos sólo se debe permitir si previamente la lista de reproducción no supera los 60 minutos, mientras que para los pagos no hay límite. Además, es muy importante que un usuario gratuito pueda pasar a ser pago en cualquier momento y viceversa. La siguiente solución funciona, pero tiene un problema fundamental. *(2 puntos)*
   a. ¿Cuál es el problema y qué consecuencias trae aparejadas?
   b. Modificar la solución resolviendo dicho problema.

```
#UsuarioPago
vi: listaReproduccion ...
>> agregarSiPuede: unElemento
  listaReproduccion agregar: unElemento

#UsuarioGratuito (hereda de UsuarioPago)
>> agregarSiPuede: unElemento
  listaReproduccion duracion <= 60
      ifTrue: [ super agregarSiPuede: unElemento ].
```