PARADIGMAS DE PROGRAMACIÓN – EXAMEN FINAL – 10/08/2013

Punto 1

Se quiere saber el sueldo de un empleado. El mismo es equivalente al importe que cobra por hora, que depende de la categoría del empleado, por la cantidad de horas que trabajó.
Si el empleado es contratado, trabaja una cierta cantidad constante de horas diarias. Esta cantidad de horas depende de cada empleado. Además se se sabe la cantidad de días que trabajó.
Si el empleado es de planta, trabaja 200 horas más las horas extras que hizo. De cada empleado se conoce esa cantidad de horas extras.

Una solución propuesta es:

```
#Contratado
	vi: categoria, cantDias, horasPorDia, datosPersonales
	>>sueldo
		^categoria importePorHora * horasPorDia * cantDias
#DePlanta
	vi: categoria, horasExtra, datosPersonales
	>>sueldo
		^categoria importePorHora * (200 + horasExtras)
```

Se asume que importePorHora está definido adecuadamente.
Desde algún otro objeto, se envía a unEmpleado (que puede ser contratado o de planta):

```
unEmpleado sueldo
```

a) ¿Funciona correctamente la solución pese a no tener herencia? Justificar.
b) Ya sea para hacer que funcione o para mejorarla, modificar la solución incorporando el concepto de herencia.
c) Surge la posibilidad que se efectivice a un empleado, es decir que deje de ser contratado y pase a ser de planta, y que se lo precarice, que es lo opuesto. Estos cambios no afectan la categoría del empleado ni sus datos personales. Si cree que es posible manteniendo lo ya hecho y agregando nuevos métodos "precarizar" y "efectivizar" con los argumentos necesarios, hacerlos. En caso contrario, realizar los cambios en la solución anterior para que sea posible, sin codificar los nuevos métodos pero justificando conceptualmente.

Punto 2

Una empresa tiene información de los clientes y sus deudas. Se quiere averiguar quién es el cliente que más debe luego de vender a crédito a todos los clientes un nuevo producto. Hay un porcentaje de descuento que depende de cada cliente. Se tiene un método que permite incrementar la deuda, que devuelve el valor actualizado de la deuda.

```
#Cliente
	vi: deuda
	>>vender: producto
		^deuda := deuda + (producto importe * (1 - self descuento))
```

Asumir que los métodos importe y descuento están resueltos correctamente.
Para resolver el problema principal, se define el siguiente método:

```
>>mayorDeudorLuegoDeVederle: producto
	^(clientes asSortedCollection: [:a :b |
		(a vender: producto) > (b vender: producto)] ) first
```

La solución es sintácticamente correcta, sin embargo tiene un problema.

a) Explicar cuál es ese problema. Justificar conceptualmente y comentar cómo encararía una nueva solución que lo resuelva.
b) ¿Qué pasaría si se estuviese en el paradigma funcional?

Punto 3

Se tienen las siguientes funciones:

```
menorSegun f a b
       | f a < f b = a
       | otherwise = b

minimoQueCumple f criterio (x:xs) = ( foldl (menorSegun f) x . filter criterio ) xs
```

1.
    - La implementación dada de minimoQueCumple requiere de los paréntesis subrayados. ¿Cuál sería el problema en caso de no estar esos paréntesis? ¿Qué error tendríamos?
    - ¿De qué tipo es la función minimoQueCumple?
2. Se quiere obtener, dada una lista de alumnos:
    - qué alumno de los que aprobaron el primer parcial tuvo la nota más baja en el segundo
    - quién tiene el menor promedio de los 2 parciales habiendo aprobado ambos.

   A los alumnos los tenemos modelados como tuplas de tres elementos con el nombre, la nota en el primer parcial y la nota en el segundo. Y tenemos las siguientes funciones:

```
listaPrueba = [(“Julia”,5,8),(“Andres”,2,9),(“Beto”,7,2),(“Nina”,4,4)]
esNotaAprobada nota = nota >= 4
promedio a b = div (a + b) 2
```

    - Resolver los dos requerimientos como **consultas** usando minimoQueCumple y expresiones lambda (**no definir más funciones**).
    - Defina una función para resolver uno de los dos requerimientos anteriores recursivamente sin usar funciones de orden superior y compare ambas soluciones en términos de declaratividad y abstracción.
3. Se tiene la siguiente implementación en Prolog para el mismo dominio:

```
alumno(julia,5,8).
alumno(andres,2,9).
alumno(beto,7,2).
alumno(nina,4,4).
promedio(Alumno,Promedio):- alumno(Alumno,N1,N2), Promedio is (N1+N2)/2.

mejorPromedio(Persona,Promedio):-
       %Armo una lista con functores promAlu/2
       findall(promAlu(Persona,Promedio), promedio(Persona,Promedio), Proms),
       not((member(promAlu(Otro,P2),Proms), P2 > Promedio)).
```

    1. Indique cuál sería la respuesta para cada una de las siguientes consultas con el código que se tiene y explique a qué se deben.

```
?- mejorPromedio(andres,5).
?- mejorPromedio(andres,Promedio).
?- mejorPromedio(Alumno,5).
?- mejorPromedio(Alumno,Promedio).
```

    2. El uso de listas en la solución propuesta es innecesario. Desarrolle una nueva solución similar a la original sin uso de listas. La misma debe ser totalmente inversible.