FINAL DE PARADIGMAS DE PROGRAMACION

Punto 1

Se cuenta con el predicado par con esta definición

```
par(N):- 0 is N mod 2.
```

donde el operador mod resuelve el resto de la división entera. O sea, el predicado par está correctamente definido.

A partir de este predicado se define el impar así:

```
impar(N):- N is N – 1, par(N).
```

y se hacen estas consultas

```
?- impar(3).
?- impar(4).
```

Indicar qué responde en cada caso, justificando su respuesta en relación con conceptos asociados al paradigma lógico.

Punto 2

Se cuenta con las siguientes clases Smalltalk:

#Alquiler	#Pelicula	#Cliente

vi: dias, pelicula, cliente
renovar: cuanto
  dias := dias + cuanto
dias: cuanto pelicula: peli cliente: cli
  dias := cuanto.
  pelicula := peli.
  cliente := cli.
dias
  ^dias
	vi: alquilerActual
	alquilateA: cli por: dias
	  | alq |
	  alquilerActual := Alquiler new.
	  // ver lo que se pide en ítem a.
	  cli alquilerActual: alquilerActual
	renovarAlquiler: dias
	  alquilerActual renovar: dias
	diasAlquilada
	  ^alquilerActual dias
		vi: alquilerActual
		alquilerActual: alquiler
		  alquilerActual := alquiler
		renovarAlquiler: dias
		  alquilerActual renovar: dias
		diasAlquiler
		  ^alquilerActual dias

y el siguiente código en un workspace:

```
rambo := Pelicula new.
pepe := Cliente new.
rambo alquilateA: pepe por: 4.
pepe renovarAlquiler: 2.
rambo renovarAlquiler: 3.
```

se pide:

a. escribir la línea de código que falta en el método alquilateA:por: de la clase Pelicula, en donde hay que asignarle al alquiler recién creado la cantidad de días, la película y el cliente, enviándole (al alquiler) el mensaje dias: pelicula: cliente:
b. indicar qué valor se obtiene si se ejecuta el código del workspace, y luego se evalúa esta línea
   ```
   rambo diasAlquilada.
   ```
   justificar relacionando con conceptos asociados al paradigma de objetos, si lo cree conveniente hacer un diagrama o dibujo.
c. idem ítem b. pero luego de ejecutar el código del workspace, la linea que se evalúa es
   ```
   pepe diasAlquiler
   ```

Punto 3

¿Qué relación tienen el hecho de que en el paradigma lógico resulta natural desarrollar software en forma declarativa, y el motor de inferencia de Prolog?

---

29 de Mayo de 2007

Punto 4

Se cuenta con las siguientes clases Smalltalk, de las que se muestra la parte significativa para la pregunta

```
#Envio
vi: transporte, kilometros
costo
   ^self costoAdministrativo + self costoCombustible
costoCombustible
    ^transporte costoCombustible: kilometros
costoAdministrativo
   "el costo administrativo de un viaje de hasta 1000 km es 20 pesos, si es de más de 1000 km, son 30 pesos"
   (kilometros <= 1000) ifTrue: [^20] ifFalse: [^30]

#Avion
vi. litrosArranque   "cuantos litros de combustible necesita para arrancar"
costoCombustible: kilometros
   | consumo |
   consumo := litrosArranque + (kilometros * 10)      " 10 litros por kilómetro más lo necesario para arrancar"
   ^consumo * 2              "el precio del combustible es 2 pesos por litro"

#Bicicleta
costoCombustible: kilometros
   ^0
```

Se pide:

a. indicar dónde se está usando polimorfismo, indicando qué objetos están involucrados.
b. indicar cuál de las clases indicadas es la que se aprovecha del polimorfismo, o sea, cuyo código es más sencillo gracias al polimorfismo, y cómo sería ese código si no se usara polimorfismo.
c. dar un ejemplo donde incorporando una nueva clase de objetos se acentúa el uso de polimorfismo encontrado en el ítem a., indicando qué condición deben cumplir los nuevos objetos para poder aprovechar el polimorfismo.
   Relacionar con la respuesta del ítem b. ¿qué pasa con la clase que se aprovecha del polimorfismo?
d. si se pide agregar envíos certificados cuyo costo administrativo es 30% superior al de los envíos comunes, ¿qué concepto/s usaría para codificarlo?
e. el analista opina que el "2" en el código del método costoCombustible: en Avion es incorrecto, porque el precio del combustible podría cambiar, aunque siempre es el mismo para todos los aviones.
   ¿qué concepto/s usaría para modificar el código de acuerdo al nuevo requerimiento?

En los puntos d. y e. brindar la codificación correspondiente, relacionándola con los conceptos que se están usando. Queda claro que no alcanza con la codificación, debe ser acompañada de la explicación para ser evaluada.

Punto 5

Se cuenta con estos ejemplos, uno en Haskell y otro en Smalltalk

```
head (filter even unaLista)
(unaCole select: [:n | n even]) first
```

La función filter recibe una función booleana F y una lista L, y devuelve la sublista de L de los elementos que verifican F. Puede codificarse así:

```
filter f [] = []
filter f (x:xs)
    | f x = x:filter xs
    | otherwise = filter xs
```

En la evaluación de uno de estos dos ejemplos aparece el concepto de evaluación diferida, mientras que en el otro no. Se pide:

a. indicar en cuál de los dos ejemplos aparece la evaluación diferida, relacionando con conceptos vistos en la materia.
b. Ejemplificando con los ejemplos, indicar ventajas de la evaluación diferida, respecto de casos que pueden cubrirse y de eficiencia.