Ejercicio 1.

En un parque de diversiones, se venden para cada juego diferentes entradas:

- Entrada simple, que permite ingresar una vez y se representa con el functor simple(Juego)
- Entrada diaria, que permite entrar todas las veces que se quiera durante un día y se representa con el functor diaria(Juego,Fecha)

A su vez se venden paquetes que consisten en una lista de entradas. Se tiene el siguiente predicado para calcular el precio de un paquete:

```
precioPaquete([],0).
precioPaquete([simple(Juego)|Resto],P):-
      precioPaquete(Resto,P2), P is 3+P2.
precioPaquete([diaria(Juego,_)|Resto],P):-
      precioPaquete(Resto,P2), P is 10+P2.
```

Se pide:

1. Se desea agregar al sistema la posibilidad de vender entradas múltiples, que se representarían con un functor multiple(Juego,CantVeces) y se cobran 2 pesos por cada vez que se puede entrar al juego si son hasta 5 veces y 1.5 pesos si son más de 5 veces. Al hacer esto se encuentra en la necesidad de modificar el predicado precioPaquete/2. *¿Cómo sería posible modificar el predicado precioPaquete/2 de forma que sirva para cualquier tipo de entrada que luego se quiera agregar? Indique el código de la versión modificada.*
2. Proponga brevemente una solución basada en el paradigma de objetos que posea la misma característica, es decir que permita agregar nuevos tipos de entrada sin tener que modificar el algoritmo para calcular el precio del paquete.
   Para ello se utilizar un diagrama en el que se vean las clases utilizadas y el código del método que calcule el precio del paquete; detallando qué se debe hacer para agregar nuevos tipos de entrada sin modificar la implementación del algoritmo.
3. Indique cuál es el concepto que aparece en ambos paradigmas y que me permite obtener ese beneficio.

Ejercicio 2

En un programa Haskell se tienen las siguientes definiciones (sqrt y atan están definidas para los números):

```
aPolares (x, y) = (sqrt (x * x + y * y), atan (y / x))
duplica x = (x, x)
```

Y además se conoce lo siguiente (del prelude):

```
fst (a, _) = a
map f [] = []
map f (x:xs) = (f x):map f xs  (o bien map f xs = [ f x | x <- xs ] )
(!!) (x:xs) 0 = x
(!!) (_:xs) n = (!!) xs (n-1)
```

Si evaluamos lo siguiente:

```
(fst.(!!4)) ((map (aPolares.duplica) [1..] ))
```

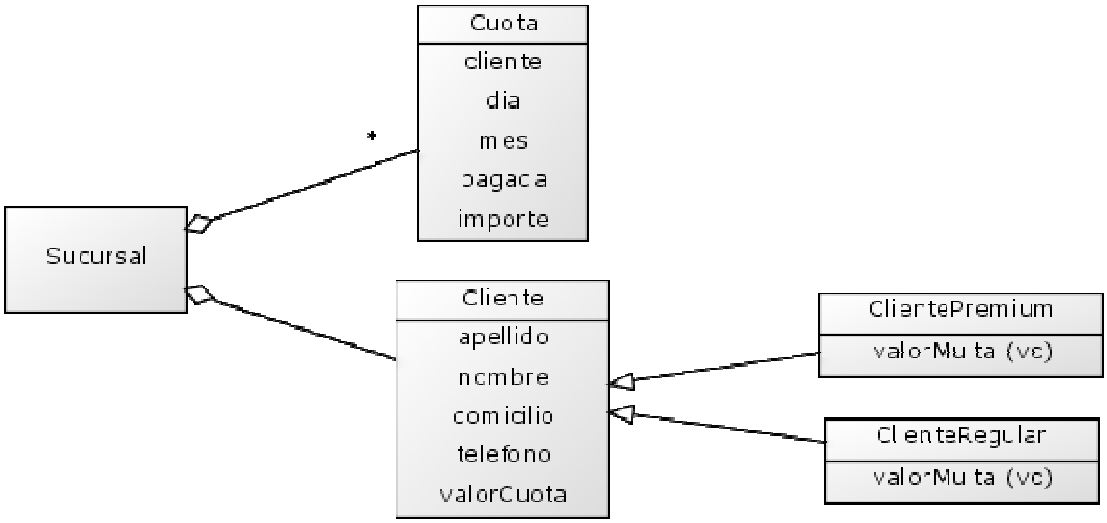
1. Indique dónde se utilizan funciones de orden superior. ¿Cómo puede identificar su uso?
2. Indique dónde se componen funciones. Realice análisis de dominio e imagen. ¿La composición está bien aplicada?
3. Si se considera la expresión **`(map (aPolares.duplica) [1..])`** por sí misma. ¿Termina la evaluación de esta expresión?. Explique y justifique el por qué de su respuesta.

Ejercicio 3

Un requerimiento de usuario (una cadena de videoclubes) dice lo siguiente:

"Tenemos 2 tipos de clientes: los clientes regulares y los clientes premium. De ambos tipos se guarda su nombre y apellido, domicilio y teléfono. Para cada tipo de cliente se tienen diferentes reglas; los cliente premium pueden retirar las películas por más tiempo, pueden retirar varias películas por vez y no se les cobra multa en caso de atrasos de hasta un día. Los clientes eventualmente pueden decidir "cambiar" de tipo, en ese caso simplemente se ajusta el plan".

La solución que le proponemos es la siguiente:




```
pagarCuota: cliente dia: dia mes: mes
| cuota|
"Busco si ya tengo la cuota generada, o
creo una"
cuota := cuotas detect: [ :c| c cliente ==
cliente and: [ c mes == mes]]
ifNone: [Cuota de:cliente para: mes].
cuota marcarPaga: dia.
```

```
cambiarTipo: cliente nuevoTipo: tipo
|c |
"Creo un cliente del tipo pedido"
c := ((tipo == 'Premium') ifTrue:
[ClientePremium] ifFalse: [ClienteNormal])
new.
c copiarDatosDe: cliente.
"Ahora hago el cambiazo"
clientes remove: cliente.
Clientes add: c.
```

Indique

1- ¿Qué problemas trae el reemplazo de un objeto por otro?

2- ¿Qué alternativas daría para resolver este problema? Realice un diagrama de clases indicando los cambios.

Ejercicio 4

El video club organiza un sorteo entre sus clientes poniendo en una urna todos los papelitos con los números de cliente y sacando tres. A ellos les da los premios grandes, mientras que a todos los demás les da un premio consuelo. El programa en Prolog que lo resuelve es lo siguiente:

```
%Acá están todos los clientes
nroCliente(juan, 1234).
nroCliente(pedro, 9999).
nroCliente(ana, 12).
nroCliente(julia, 100).
```

```
nroCliente(oscar, 7711).
 %Estos son los premios
premioGrande(1, "viaje a Bariloche").
premioGrande(2, "heladera").
premioGrande(3, "radio FM").
premioConsuelo("llavero").
%Y estas las reglas
gano(Cliente, Premio):- nroCliente(Cliente, Numero),
      sorteo(Numero, Posicion), premioGrande(Posición, Premio).
gano(Cliente, Premio):- nroCliente(Cliente,Numero), premioConsuelo(Premio).
%Cuando se hizo el sorteo, se agregaron estos hechos
sorteo(1234, 1).
sorteo(9999, 2).
sorteo(7711, 3).
```

Lo que esperamos al ejecutar las siguientes consultas es lo siguiente:

`?- gano(juan, X).`
`X = "viaje a Bariloche";`
`no`
	`?- gano(julia, X).`
	`X = "llavero";`
	`no`

1- ¿Es correcta la solución? En caso afirmativo, justificar los principales conceptos que se usaron; en caso contrario, explicar el motivo y corregirlo.

2- Se quiere hacer lo mismo en Haskell y se construye la siguiente solución:

```
gano x | numeroSorteo >= 1 && numeroSorteo <= 3 = premio numeroSorteo
       | otherwise = premioConsuelo
      where numeroSorteo = sorteo (buscarNumeroCliente x)
```

La función **`buscarNumeroCliente`**, recibe el nombre de cliente y devuelve su número. Ya está hecha y funciona bien.

```
-- Estas son las funciones que devuelven los premios
premioGrande 1 = "viaje a Bariloche"
premioGrande 2 = "heladera"
premioGrande 3 = "radio FM"
premioConsuelo = "llavero"
sorteo 1234 = 1
sorteo 9999 = 2
sorteo 7711 = 3
```

También se espera poder ejecutar lo siguiente:

`?- gano "juan"` `"viaje a Bariloche"`	`gano "julia"` `"llavero"`

¿Funciona correctamente? En caso afirmativo, justificar los principales conceptos que se usaron; en caso contrario, explicar el motivo y corregirlo.

3- Identifique y dé ejemplos de los diferentes tipos de consulta que puede realizar en una y otra versión.

4- Hay un cambio en la forma de identificar a los clientes. Analizar los dos casos siguientes:
   a. El código en vez de ser de tipo numérico pasa a ser una tupla (tipo, numero) para todos los clientes.
   b. El código cambia sólo para los nuevos clientes, los viejos siguen manteniendo el número.

   ¿Qué pasa en cada caso con las soluciones planteadas? ¿Es posible adaptarlas al nuevo requerimiento? Justifique.