Condiciones de aprobación:

Para aprobar es necesario simultáneamente: • obtener 8 puntos de 14, y • obtener al menos la mitad de los puntos en cada paradigma.	Y recordá: en todas tus respuestas sé puntual, no pierdas el foco de lo que se pregunta. Respuestas en exceso generales son tan malas como respuestas incompletas.

Parte A

Dada la siguiente función:

```
func [ ] xs  _ = []
func xs [ ]  _ = []
func (x:xs) (y:ys) z = z x y : func xs ys z
```

1. Dado el siguiente dominio e imagen propuesto para la función, ¿encuentra algún error? Si es incorrecto, corríjalo y justifique.
   ```
   func :: [a] -> [b] -> (a-> b) -> [b]
   ```
2. Considerando este ejemplo.
   a) Además de la recursividad, ¿qué otros conceptos importantes se están utilizando en la definición de la función func? Mencionar al menos 2 y dónde es que esto se pone en evidencia.
   b) Dada una lista de artículos escolares con sus cantidades con este formato (nombreArticulo, cantidad)
   ```
   cantidades = [("regla", 20),("goma", 30),("lapicera", 40)]
   ```
   Y una lista de precios por unidad de los mismos artículos en el mismo orden con este formato (nombreArtículo, precioUnidad)
   ```
   precioUnidad = [("regla", 40),("goma", 15),("lapicera", 20)]
   ```
   1. Aplicar la función del punto 1 para obtener una lista de tuplas con este formato: (nombreArticulo, cantidad, precioUnidad)
   2. Aplicar la función del punto 1 para obtener una lista de tuplas con este formato: (nombreArticulo, precioTotal)
3. Reescribir la solución sin utilizar recursividad sabiendo que existen las siguientes funciones que podrían ser de utilidad:
   ```
   zip :: [a] -> [b] -> [(a, b)]
   zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
   ```

Parte B

```
comproEn(ale, soda, wallmart).
comproEn(ale, torta, wallmart).
comproEn(bart, bowl, wallmart).
comproEn(bart, soda, wallmart).
comproEn(bart, soda, superDia).
comproEn(bart, torta, superDia).
```

```
vende(wallmart,soda).
vende(wallmart,torta).
vende(wallmart,bowl).
vende(superDia,torta).
vende(superDia,soda).
```

```
cumpleCondicion(Cliente,Lugar) :-
    findall(Producto, vende(Lugar,Producto), ProductosVendidos),
    findall(Producto, comproEn(Cliente,Lugar,Producto), ProductosComprados),
    incluye(ProductosComprados, ProductosVendidos)[1].
```

`incluye/2` es un predicado que es cierto cuando los elementos de la segunda lista están incluidos en la primera

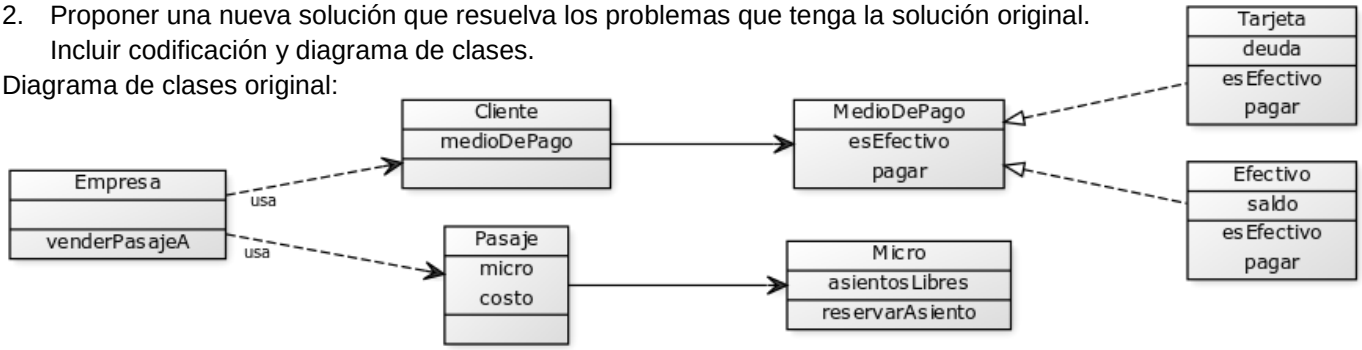
1. Descubriendo qué sucede:
   a. ¿Cuál es el valor de verdad de la siguiente consulta?
      ```
      ?- cumpleCondicion(ale,wallmart).
      ```
   b. Renombrar el predicado cumpleCondicion/2 para que exprese claramente cuál es su función.
2. Analizar la inversibilidad del predicado `cumpleCondicion/2` indicando brevemente qué responde ante las siguientes consultas y por qué:
   a. `?- cumpleCondicion(bart,Local).`
   b. `?- cumpleCondicion(Cliente,wallmart).`
   c. `?- cumpleCondicion(Cliente,Local).`
3. Desarrollar una solución más declarativa para el predicado cumpleCondicion/2 de modo que no sea necesario el uso de listas. La misma debe ser completamente inversible.

Parte C

La empresa Arrancar.com vende pasajes de micro a varias regiones del país. Se desea implementar una parte del proceso de venta de un pasaje a un cliente. Al vender un pasaje, se le cobra al cliente lo correspondiente de acuerdo al costo del pasaje (hay un 10% de descuento al pagar en efectivo) y se reserva un asiento en el micro (que pasa a tener un asiento libre menos). Además, pagar con tarjeta aumenta la deuda de la tarjeta, y pagar con efectivo disminuye el dinero, siempre que haya el suficiente. Tampoco debería venderse el pasaje si el micro está lleno.

1. Responder Verdadero o Falso y justificar conceptualmente en todos los casos:
    a. Es correcto que la responsabilidad de venta de pasajes con el cálculo de precio final esté en el objeto empresa, porque en la realidad es la empresa la que decide cuánto debería cobrarse.
    b. Como ambos medios de pago (tarjeta y efectivo) entienden el mensaje esEfectivo, hay un buen uso del polimorfismo.
    c. Para evitar que se reserve un asiento cuando no hay suficiente dinero en efectivo, la operación para el pago del pasaje debería retornar un código de error. De esa forma se puede chequear si debería reservarse o no el pasaje.
    d. La solución tiene problemas relacionados con encapsulamiento.
2. Proponer una nueva solución que resuelva los problemas que tenga la solución original. Incluir codificación y diagrama de clases.

Diagrama de clases original:



Código original del requerimiento en Smalltalk y Wollok (asumir que además existen los getters y setters usados y las clases que se muestran en el diagrama anterior):

```
#Empresa>>vender: pasaje a: cliente
 pasaje micro asientosLibres > 0 ifTrue: [
   cliente medioDePago esEfectivo ifTrue: [
      cliente medioDePago pagar: pasaje costo * 0.9
    ] ifFalse: [
      cliente medioDePago pagar: pasaje costo
    ].
   pasaje micro reservarAsiento.
 ].

#Micro>>reservarAsiento
 asientosLibres := asientosLibres - 1.
```

```
#Tarjeta>>esEfectivo
    ^ false
#Tarjeta>>pagar: monto
   deuda := deuda + monto.

#Efectivo>>esEfectivo
    ^ true
#Efectivo>>pagar: monto
 dinero >= monto ifTrue: [
  dinero := dinero - monto.
 ].
```

```
object empresa {
 method venderPasajeA(pasaje, cliente){
  if(pasaje.micro().asientosLibres()>0){
   if(cliente.medioDePago().esEfectivo()){
    cliente.medioDePago().pagar(pasaje.costo()*0.9)
   } else {
     cliente.medioDePago().pagar(pasaje.costo())
   }
   pasaje.micro().reservarAsiento()
  }
 }
}
class Micro {
  var asientosLibres
  method reservarAsiento(){
     asientosLibres -= 1
  }
}
```

```
class Tarjeta {
  var deuda = 0
  method esEfectivo(){
    return false
  }
  method pagar(monto){
    deuda += monto
  }
}
class Efectivo{
  var dinero
  method esEfectivo(){
    return true
  }
  method pagar(monto){
    if(dinero >= monto){
     dinero -= monto
    }
  }
}
```