




## Condiciones de aprobación

Para aprobar es necesario simultáneamente: <ul style="list-style-type: none"> <li>completar el 60% del examen, y</li> <li>obtener al menos la mitad de los puntos <b>en cada paradigma</b>.</li> </ul>	<b>En todas tus respuestas sé puntual</b> , no pierdas el foco de lo que se pregunta. Respuestas en exceso generales son tan malas como respuestas incompletas. 
--	---

## Parte A

Una empresa de venta online necesita desarrollar parte de su sistema de ventas. Sabemos que a los clientes VIP les hacen 25% de descuento en cada compra, y a los demás sólo les hacen un 10% de descuento si es la primera vez que compran. Además es necesario manejar el stock del producto comprado, de modo que no se venda un producto que no hay disponible. Lógicamente, no debería venderse el producto a un cliente que no tenga suficiente dinero para pagarlo. En cualquier momento un cliente debería poder solicitar un cambio de plan.

Se desarrolló la siguiente implementación de este requerimiento en objetos. El código no incluye declaración de atributos, getters ni setters; si se usan, puede asumirse que existen.

 <pre> #Empresa&gt;&gt;vender: producto     a: cliente     producto hayStock ifFalse: [^ false ].     cliente plan esVip ifTrue: [         cliente pagar: producto costo * 0.75     ] ifFalse: [         cliente yaCompro not ifTrue: [             cliente pagar: producto costo * 0.9         ] ifFalse: [             cliente pagar: producto costo         ]     ].     cliente agregarProducto: producto.     producto decrementarStock.     ^ true.  #Producto&gt;&gt;decrementarStock     stock := stock- 1. #Producto&gt;&gt;hayStock     ^ stock &gt;= 1.  #PlanVip&gt;&gt;esVip     ^ true  #PlanNormal&gt;&gt;esVip     ^ false  #Cliente&gt;&gt;pagar: monto     dinero &gt;= monto ifTrue: [         dinero := dinero - monto.     ]. #Cliente&gt;&gt;yaCompro     ^ productos notEmpty #Cliente&gt;&gt;agregarProducto: producto     ^ productos add: producto </pre>	 <pre> class Empresa {     method venderA(producto, cliente){         if(not producto.hayStock()){ return false }         if(cliente.plan().esVip()){             cliente.pagar(producto.costo()*0.75)         } else {             if(not cliente.yaCompro()){                 cliente.pagar(producto.costo()*0.9)             } else {                 cliente.pagar(producto.costo())             }         }         cliente.agregarProducto(producto)         producto.decrementarStock()         return true     } }  class Producto{     method decrementarStock(){ stock -= 1 }     method hayStock(){ return stock &gt;= 1 } }  class PlanVip {     method esVip(){ return true } }  class PlanNormal {     method esVip(){ return false } }  class Cliente {     method pagar(monto) {         if(dinero &gt;= monto){ dinero -= monto }     }     method yaCompro(){ return not productos.isEmpty() }     method agregarProducto(producto){         productos.add(producto)     } } </pre>
--	--

- Para las siguientes afirmaciones relativas a esta solución, responder Verdadero o Falso y justificar conceptualmente en todos los casos:
  - Es correcto que sea la empresa quien centralice toda la lógica de venta de un producto, porque representa fielmente la realidad.
  - Como ambos planes entienden el mensaje *esVip*, hay un buen uso del polimorfismo entre ellos.

- c. Por cómo se resolvió este requerimiento, no será un problema si se desea que un cliente cambie de plan.
- d. No está bien la lógica de venta, ya que si no se podía pagar el producto, no se le va a cobrar nada, pero el stock se va a decrementar igual y se va a agregar a los productos del cliente. Lo correcto sería que el método *pagar* retorne un booleano para saber si el producto fue abonado.

2. Proponer una solución superadora de los problemas detectados. Implementar codificación y diagrama de clases.

## Parte B

A partir del siguiente código:

```
data Futbolista = UnFutbolista {posiciones :: [Int], goles :: Int} deriving (Show, Eq)
tito = UnFutbolista [9,10] 47
meterGoles cant (UnFutbolista posiciones goles) = UnFutbolista posiciones (goles+cant)
jugarDe posicion (UnFutbolista ps goles) = UnFutbolista (agregarPosicion posicion ps) goles
agregarPosicion p ps | not (elem p ps) && between 1 11 p = p : ps
                    | otherwise = ps
```

1. En cada una de las consolas siguientes, ¿qué sucede en la última consulta? ¿Hay respuesta? ¿Cuál es? **Justifique conceptualmente** en todos los casos.

<p>Consola A)</p> <pre>&gt; tito UnFutbolista [9,10] 47  &gt; meterGoles 2 tito UnFutbolista [9,10] 49  &gt; goles tito ???????</pre>	<p>Consola B)</p> <pre>&gt; jugarDe 2 (UnFutbolista [3..] 1) ???????</pre>	<p>Consola C)</p> <pre>&gt; meterGoles 3 ???????</pre>
---	--	--

2. Usando composición, escriba una consulta que permita conocer si, luego de haber jugado de 4 y haber jugado de nuevo de 10, es cierto que tito jugó en más de 2 posiciones.
3. Dadas las funciones *f*, *g* y *h* de las cuales sólo conocemos su tipo:  
 $f :: (a \rightarrow b) \rightarrow c \rightarrow (b, c)$        $g :: a \rightarrow a \rightarrow \text{Bool}$        $h :: [a] \rightarrow [b]$   
Indicar cuáles de las siguientes expresiones tipan (indicando cuál es el tipo de dicha expresión) y cuáles no (indicando el motivo por el cual no tipan):

- a.  $f \ h \ . \ g \ 3$
- b.  $g \ 3 \ . \ h$
- c.  $f \ even \ . \ g \ 1 \ 2$

## Parte C

Dada la siguiente base de conocimientos:

<pre>% Relaciona un alumno con un final al que se anotó. anotado(ana,paradigmas,25). anotado(ana,fisicaII,9). anotado(beto,paradigmas,25). anotado(camilo,paradigmas,25).</pre>	<pre>fecha(paradigmas,11). fecha(paradigmas,18). fecha(paradigmas,25). fecha(fisicaII, 9). fecha(fisicaII,16). fecha(fisicaII,23).</pre>
<pre>ultimaFecha(Materia,Fecha):- findall(Dia, fecha(Materia, Dia), Fechas),                              max_list(Fechas, Fecha).</pre>	
<pre>% El predicado max_list/2 relaciona a una lista con su elemento máximo.</pre>	

1. Con la solución dada, ¿qué se obtendrá como respuesta a la consulta `ultimaFecha(Materia,Fecha)`? Explicar cómo se llega a esa conclusión.
2. Reescribir el predicado `ultimaFecha` sin usar listas y comparar ambas soluciones en términos de declaratividad. Asegurar que la nueva solución sea inversible.
3. Asumiendo que el predicado `ultimaFecha` fue corregido como se solicitó, responder para las siguientes consultas qué significado tiene y qué soluciones genera Prolog.
  - a. `?- forall(anotado(_,paradigmas,Fecha), ultimaFecha(paradigmas,Fecha)).`
  - b. `?- anotado(Alumno1, Materia, Fecha), anotado(Alumno2, Materia, Fecha), Alumno1 \= Alumno2.`