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

La información de stock de productos naturales se modela mediante listas en Haskell, en las que se indica el nombre del producto y la cantidad en stock, por ejemplo:

```
[("manzana", 50), ("banana", 30), ("naranja", 40), ("papa", 40), ("tomate", 25)]
```

1. A diferentes desarrolladores se les pidió que hicieran una función que calcule la cantidad de unidades de todos los productos de una lista. Se consiguió lo siguiente:

```
cantidadTotal1 productos = sum (map snd productos)
cantidadTotal2 productos =  (sum.map ((_,c)->c)) productos
cantidadTotal3 = sum.map snd
cantidadTotal4 productos = foldl (+) 0 (map snd productos)
cantidadTotal5 [ ] = 0
cantidadTotal5 ((_,cantidad):productos) = cantidad + cantidadTotal5 productos
```

   Indicar si hay alguna que no funcione correctamente, marcando los errores cometidos.
   Entre las que funcionen bien, determinar la/s que te parezcan mejores y la/s que creas que son peores. Justificar conceptualmente las elecciones.

2. Se define la siguiente función:

   ```
   muchasManzanas = ("manzana", 50):muchasManzanas
   ```

   Qué sucede al hacer la siguiente consulta (con cualquiera de las anteriores)

   ```
   cantidadTotal muchasManzanas
   ```

3. Se tiene la siguiente función:

```
nombresFrutasPesadas pesoMaximo productos = (map fst.funcionMagica pesoMaximo) productos
```

   Definir *funcionMagica* para que la función principal haga lo que su nombre sugiere

Parte B

Un sistema debe gestionar reservas de productos solicitados por los clientes. El sistema sólo permite realizar reservas mientras haya cantidad de stock disponible. Para esto, se tiene el siguiente modelo:

```
#Cliente
reservar: unaCantidad de: unProducto
  |unaReserva|
  (self disponibleDe: unProducto) >=
unaCantidad ifTrue: [
       unaReserva := Reserva new.
       unaReserva cantidad: unaCantidad.
       unaReserva producto: unProducto.
       unaReserva cliente: self.
       Reserva reservas add: unaReserva ]

#Producto (VI: stock, nombre)

(se asume que los getters y setters están
definidos correctamente)
```

```
#Cliente
disponibleDe: unProducto
  ^unProducto stock -
    (Reserva cantidadReservadaDe: unProducto)

#Reserva (VI: cantidad, producto, cliente; VC:
Reservas)
cantidadReservadaDe: unProducto (MC)
  |suma|  suma := 0.
  Reservas do: [:unaReserva |
    unaReserva producto nombre = unProducto
nombre ifTrue:
      [suma := suma + unaReserva cantidad]].
  ^suma
```

1) La solución presentada tiene algunas cuestiones a analizar:
    a) ¿Qué te parece la delegación de responsabilidades a los clientes? ¿Es correcto que cada cliente se encargue de hacer sus reservas? ¿Y que calcule la cantidad disponible del producto que quiere reservar? Justificá, y en caso negativo planteá qué objeto debería realizar cada tarea.
    b) La clase Reserva, como su nombre sugiere, se encarga de todo lo referente a las reservas: tiene los atributos propios de cada reserva como variables de instancia y la colección de reservas con todas las reservas realizadas como variable de clase para que sea única. Esta decisión permite que el sistema funcione, pero tiene problemas conceptuales. Explicalos y proponé una mejor solución.
    c) La suma de las cantidades de las reservas presenta detalles algorítmicos que pueden simplificarse. Corregilo y justificá, mencionando el concepto asociado.
    d) Para el caso en que no haya suficiente stock disponible, ¿estás de acuerdo con el comportamiento dado por la solución adoptada? Justificar.
    e) ¿Qué opinas de usar el nombre del producto como criterio para buscar las reservas? Si no estás de acuerdo, proponé una mejora y justifica conceptualmente.
2) Llega una aclaración sobre cómo debe funcionar el módulo: Se trata de reservas de productos naturales como el ejercicio anterior, donde entre otros productos, hay frutas. Cuando se hace una reserva de frutas, en realidad se debe reservar un 50% más de lo solicitado, dado que tiende a pudrirse más rápido que los demás productos y hay que guardar algo extra, por las dudas. Realizá los cambios que sean necesarios en la solución propuesta para contemplar el requerimiento y explicar el concepto que se utiliza para que el impacto en la solución sea mínimo.

Parte C

En la base de conocimiento hay cierta información de cada artesano, se conocen los diferentes tipos de artesanías que existen y el material característico de cada uno. Lo más importante, qué tipos de artesanías hace cada artesano:

```
artesano(carlos, cordoba, 2004).
artesano(maria, bariloche, 2000).
artesano(elsa, humahuaca, 1977).
artesania(ceramica, arcilla).
artesania(plateria, metales).
artesania(tejido, lana).
artesania(estampado, tela).
```

```
hace(carlos, ceramica).
hace(carlos, plateria).
hace(maria, ceramica).
hace(elsa, tejido).
hace(elsa, estampado).
```

Se tienen tres versiones de un predicado que cuenta:

```
a)
cuantos( Persona, C):-
  findall( A, hace(Persona, A),
Lista),
  length(Lista, C).
```

```
b)
cuantos( Persona, C):-
  artesano(Persona, _,_),
  findall( A, hace(Persona, A),
Lista),
  length(Lista, C).
```

```
c)
cuantos( Persona, C):-
  artesano(Persona, _,_),
  artesania(A,_),
  findall( A, hace(Persona, A),
Lista),
  length(Lista, C).
```

1. ¿Qué se obtiene en cada caso ante la consulta `cuantos(Alquien, Cant).`? Justificar conceptualmente las diferencias
2. Se agrega a la base de conocimientos un predicado hayMaterial/2 que relaciona a cada lugar con cada material que puede conseguirse allí. Por ejemplo:
   `hayMaterial(cordoba,arcilla).`
   Desarrollar un predicado malUbicado/1 que permita conocer a los artesanos que no tengan en su lugar ninguno de los materiales principales que necesitan para sus oficios (no se puede usar listas).