PUNTO 1)

Tenemos las siguientes clases

```
Factura
v.i. items, fecha
importe
   ^items inject: 0 into: [:suma :item | suma + item importe]
cantidadDeProductos
   ^items inject: 0 into: [:suma :item | suma + item cantidad]

FacturaConDescuento (subclase de Factura)
v.i. porcentajeDescuento
importeDescontado
   ^(super importe * porcentajeDescuento) / 100
importe
   | descuento |
   descuento := self importeDescontado.
   "si el descuento es menos que un peso, no se tiene en cuenta"
   (descuento < 1)
      ifTrue: [^super importe]
      ifFalse: [^super importe - self importeDescontado]

FacturaAPrecioFijo (subclase de Factura)
v.i. precioPactado
importe
   ^precioPactado
esGrosa
   ^self cantidadDeProductos > CantidadMinimaGrosa
```

Se pide

a. Indicar por qué en el método `importeDescontado` de la clase `FacturaConDescuento` dice `super importe` y no `self importe`. Explicar qué tienen de similar y qué tienen de distinto `super` y `self` señalando similitud/es y diferencia/s en este caso concreto.

b. En el método `esGrosa` de la clase `FacturaAPrecioFijo`, como están ahora las clases, poner `super` en vez de `self` delante de `cantidadDeProductos` no cambia el resultado que se obtiene al enviar el mensaje correspondiente a una instancia. En cambio, si se agrega un método en una de estas tres clases, se produce una diferencia.
Indicar qué método hay que agregar en qué clase para provocar dicha diferencia (no hace falta hacer el código) y explicar por qué se produce.

c. `CantidadMinimaGrosa` es variable de clase.
Indicar bajo qué condiciones no está bien usar una variable de clase y debe ser de instancia. En caso de que no se den esas condiciones (o sea, está bien que la variable sea de clase) qué ventaja/s tiene que la variable sea de clase en lugar de instancia.

PUNTO 2)

Agregamos clientes al modelo anterior, queremos obtener de un cliente lo que vamos a llamar su "muestra simpática", que es el conjunto de sus 5 primeras facturas simpáticas.
Decimos que una factura es simpática si se hizo en diciembre.
Se le dio a este requerimiento la implementación que sigue:

```
Cliente
v.i. facturas

muestraSimpatica
```

```
    ^(facturas select: [:fact | fact esSimpatica])
     copyFrom: 1 to: 5

Clase Factura
esSimpatica
  ^fecha monthIndex = 12
```

Se implementa esto mismo en Prolog, en un pseudo-Pascal y también en Haskell.

En Prolog tenemos un functor factura(Fecha,Items), y los predicados

- monthIndex(Fecha,Mes)
- seFacturo(Cliente,Factura)
- take(Lista,Ene,LosPrimerosNDeLaLista)

Se define este predicado

```
muestraSimpatica(Cli,MS):- esCliente(Cli),
   findall(Fact,
      (seFacturo(Cli,Fact), esSimpatica(Fact)),
      FS), take(FS,5,MS).

  esSimpatica(factura(Fecha,_)):- monthIndex(Fecha,12).
```

En Haskell una factura está representada por una tupla (Fecha, Items), tenemos las funciones

- facturas cliente, devuelve una lista de facturas
- take lista cant

y definimos esta función

```
muestraSimpatica cliente = take 5
   ( filter ((12==).monthIndex.fecha) (facturas cliente) )
```

En el pseudoPascal representamos cada factura mediante un registro con los campos fecha e items. Tenemos las funciones

- facturas(cliente), devuelve un vector de facturas
- monthIndex(fecha)

Implementamos esta función

```
muestraSimpatica (cliente)
  Factura[5] muestra;
  Factura[1000] lasFacturas = facturas(cliente);
  int i,j = 0;
  while (i < 1000) {
    if (monthIndex(lasFacturas[i].fecha) = 12) {
      muestra[j++] = lasFacturas[i];
      if (j == 5) { break };
    }
    i++;
  }
  return muestra;
```

Se pide

a. si se da cierta condición, la ejecución de dos de estas cuatro implementaciones va a ser notablemente más rápida que las otras dos, por razones relacionadas con conceptos vistos en la materia.
   Indicar. cuál es esta condición, cuáles son las dos implementaciones que serían más rápidas, y con qué concepto está relacionada esta mayor performance en cada caso (ayuda: son distintos conceptos en los dos casos).

~~Llegar a alguna conclusión acerca de la diferencia de enfoque entre los dos paradigmas "rápidos" respecto de cómo se logra esta velocidad.~~

b. En una de las cuatro implementaciones, además de consultar por la muestra simpática para un cliente determinado, puedo hacer otra consulta a partir de la implementación que detallamos. Indicar en qué implementación pasa esto, qué otra consulta puedo hacer, y con qué concepto está relacionada esta posibilidad adicional.

c. En las implementaciones de Haskell y Prolog, para obtener el conjunto de facturas simpáticas a partir del conjunto de facturas se usa un concepto que no aparece tan naturalmente en los otros dos paradigmas. Indicar cuál es ese concepto, y señalar su uso en las implementaciones indicadas.

d. En la implementación de Haskell, indicar qué reciben y qué devuelven las funciones fecha y monthIndex; y cómo lo puede deducir a partir de la definición de la función `muestraSimpatica`, relacionando con conceptos vistos en la materia.

e. Hay un cambio en los requerimientos: para las facturas a precio fijo la condición de ser simpática no tiene que ver con la fecha, sino que se considera a una factura a precio fijo simpática si tiene exactamente 9 ítems.
Para agregar este cambio en la implementación de objetos respetando las ideas del paradigma, hay que hacer algún cambio. Indicar qué cambio hay que hacer, y con qué conceptos vistos en la materia está relacionado el cambio propuesto.

f. De las soluciones de Haskell, Prolog y seudoPascal , hay una en la que es más sencillo hacer el mismo cambio del punto anterior. Indicar cuál es y explicar cómo sería el cambio (no hace falta detallar la implementación) comparando con el cambio hecho en Smalltalk. Dar una explicación de por qué en las otras no se puede hacer tan fácil lo mismo.

PUNTO 3)

Sobre el mismo ejemplo, nos piden la cantidad de productos de los dos primeros ítems de una factura. Lo que sigue es una implementación en Smalltalk y otra en Prolog. En prolog, se asume que la factura tiene como ítems a un lista de functores donde se escpecifica el importe y la cantidad:

```
Factura
  cantItems1y2
    |n|
    n := (items at:1) cantidad.
    n := n + (items at: 2) cantidad.
    ^n
```

```
cantItems1y2(factura(Fact,Items),Cant):-
   Cant = 0,
   nth1(Items, 1, item(_,C1)), Cant = Cant + C1,
   nth1(Items, 2, item(_,C2)), Cant = Cant + C2.
```

Se pide

a. Una de estas implementaciones anda bien, la otra no. Indicar cuál es la que no anda bien, señalar el problema en el código, y relacionarlo con conceptos vistos en la materia.

b. Si en Smalltalk y en Prolog pregunto la `cantItems1y2` del string “hola”
`“hola” cantItems1y2` `?- cantItems1y2(“hola”,Cantidad).`
Indicar la reacción en cada caso, y con qué concepto está relacionada la diferencia.