PUNTO 1)

Tenemos las siguientes clases

```
Cliente
v.i.: ventas
ventaMasAlta
   ^(ventas asSortedCollection:
           [:v1 :v2 | v1 importe > v2 importe]) first
agregarVenta: venta
   ventas add: venta

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]
esSimpatica
   "las facturas simpaticas son aquellas hechas en diciembre"
   ^fecha monthIndex = 12

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

VentaDeServiciosDeExperto          Experto
v.i. experto, horas                precioPorHora
importe                               "no importa la implementación"
   ^experto precioPorHora * horas

Y este workspace
// creamos venta1 que es una FacturaConDescuento,
// creamos venta2 que es una FacturaAPrecioFijo
// creamos venta3 que es una VentaDeServiciosDeExperto
pepe := Cliente new.
pepe agregarVenta: venta1.
pepe agregarVenta: venta2.
pepe agregarVenta: venta3.

"pregunto cosas"
pepe ventaMasAlta.
venta2 importe.
venta1 esSimpatica.
```

Se pide

a. Completar la siguiente frase usando la palabra "mensaje", y usando no más de 30 palabras
   "un método es ..."
   Aplicar la definición al método ventaMasAlta de la clase Cliente, p.ej. relacionando con el workspace del ejemplo.
b. Completar la siguiente frase usando alguna de las palabras "método" o "métodos", y usando no más de 40 palabras
   "los mensajes que entiende un objeto son aquellos para los que hay ..."
   ¿Qué otro/s concepto/s visto/s en la materia está/n relacionado con la frase que escribiste?
   Aplicar la definición a los mensajes que se envían en el workspace del ejemplo.
c. Cuando se envía un mensaje, ¿corresponde siempre el mismo método o depende? Si depende, ¿de qué depende? ¿Qué otro/s concepto/s visto/s en la materia aparece/n?
   Aplicar la respuesta a algún envío de mensaje en el código del ejemplo donde quede claro lo que estás respondiendo.
d. ¿Qué diferencia hay entre una variable local y una variable de instancia? Aplicar la respuesta a una variable local y una de instancia en el ejemplo.

PUNTO 2)

Agregamos lo siguiente al ejemplo del punto 1

```
Cliente
v.i. (agregamos) ventasGrandes

importeTotalVendido
  | result |
  result := 0. ventasGrandes := Set new.
  ventas do:
    [ :venta | result = result + venta importe.
               (venta importe > 1000)
                   ifTrue: [ventasGrandes add: venta ].
    ].
  ^result

ventasGrandes
  ^ ventasGrandes
```

Ejecuto estas cuatro líneas de workspace en el orden en el que están

```
cuantaGuita1 := cliente1 importeTotalVendido.
ventasGrosas1 := cliente1 ventasGrandes.
ventasGrosas2 := cliente2 ventasGrandes.
cuantaGuita2 := cliente2 importeTotalVendido.
```

Es probable que una de las cuatro variables quede asignada con un valor incorrecto.
Se pide

a. indicar con qué concepto/s que vimos en la materia está relacionado el problema, y dónde aparece/n concretamente en el código presentado.
b. arreglar el código para que en el resultado de la ejecución del workspace sea correcta sin importar en qué orden ejecuto las líneas, relacionando el arreglo con el/los concepto/s indicado/s en el ítem anterior.

PUNTO 3)

Sobre el mismo ejemplo:

```
Cliente
  ventasQueCumplen: condicion
    ^ventas select: condicion
```

ejemplo de uso en el workspace

```
cliente1 ventasQueCumplen: [:venta | venta importe < 100]
```

También tenemos, sin importar la implementación

```
Factura
  estaPendiente "devuelve un booleano"
VentaDeServiciosDeExperto
  estaPendiente "devuelve un booleano"
```

En Haskell tenemos esto

```
ventasQueCumplen cliente condicion =
      filter condicion (ventas cliente)
```

más las siguientes funciones de las cuales no importa la implementación, se indica lo que devuelve cada una

```
ventas cliente          "lista de ventas"
estaPendiente venta     "booleano"
```

Estas dos expresiones tienen el mismo significado

```
head ( ventasQueCumplen cliente estaPendiente )

(cliente1 ventasQueCumplen: [:venta | venta estaPendiente])
first
```

Responder lo siguiente:

a. en el ejemplo de objetos, el método ventasQueCumplen: espera un bloque de código como parámetro; con eso se logra cierto efecto.
   Indicar qué concepto visto en la materia usamos para lograr el mismo efecto en el ejemplo de funcional, señalando el uso de ese concepto en el código presentado.
b. imaginemos un cliente que tiene 1000000 de facturas que están todas pendientes. ¿Qué puede decir del tiempo de respuesta ante la ejecución de las expresiones finales en Haskell y en Smalltalk? Dar una explicación en la que se use al menos un concepto visto en la materia.

PUNTO 4)

Siguiendo con el mismo ejemplo en objetos, supongamos este agregado

```
Cliente
  comproMuchoDe: producto  "devuelve un booleano"
```

y en una implementación de lo mismo en lógico tengo el predicado

```
comproMuchoDe(Cliente,Producto)
```

estas expresiones dan el mismo resultado

```
?- comproMuchoDe(cliente1,producto1).
cliente1 comproMuchoDe: producto1
```

Mostrar ejemplos de consultas que probablemente pueda hacer en Prolog usando solamente el predicado comproMucho que no puedo hacer en un workspace así de fácil, indicando qué conceptos vistos en la materia se están aprovechando.