Punto 1

Dada una aplicación en Smalltalk de distribución de encomiendas entre sucursales de una empresa de correos, donde se manejan distintos tipos de transporte. Lo que sigue es parte del código

```
#Sucursal(v.i.: transportes, transporteElegido)
hayTransportePara: unEnvio
    | transportesPosibles |
    transportesPosibles :=
            transportes select: [:t | t puedeLlevar: unEnvio].
    (transportesPosibles isEmpty)
            ifFalse: [transporteElegido := transportesPosibles anyOne].
    ^transportesPosibles isEmpty not

cargar: unEnvio
    transporteElegido agregarEnvio: unEnvio
```

Se pide

1. Dado el siguiente Workspace:

```
sucTucuman := Sucursal new.
envioDeTomates := Envio new.
// configuro sucTucuman y envioDeTomates
sucTucuman cargar: envioDeTomates
```

   - ¿Qué problema hay al evaluar esos mensajes?
   - ¿Cómo solucionaría este inconveniente modificando los métodos anteriores, para que el workspace funcione correctamente sin modificaciones?
   - Relacione la modificación hecha en el punto anterior con el concepto de efecto colateral (o efecto de lado).

2. Tengo las clases Camion, Barco, Moto, Combi y Avion para representan los transportes, y Envio para representar los envíos. Las condiciones para ver si un transporte puede llevar un envío son:
   - que dé el peso, la condición para esto es la misma para todos los transportes.
   - si el envío es intercontinental, no se puede llevar por tierra.
   - si el envío es gaseoso, no se puede llevar en avión.
   - si el envío es frágil, no se puede llevar por barco.

   Lo que hay que hacer cuando a un transporte se le agrega un envío es lo mismo para todos los transportes. Suponer que hay dos programadores: Juan se encarga de codificar los transportes, y Ana los envíos. La idea es que cada uno sepa lo menos posible de los detalles del trabajo del otro. Suponiendo que esta es toda la información relevante, indicar:
   a. qué clases le conviene definir a Juan, en cuál/es hay que incluir un método puedeLlevar:, y en cuál/es conviene definir un método agregarEnvio:, de forma tal que no sea necesario duplicar código. Justificar las decisiones que se toman, indicar qué conceptos se usan y dónde/cómo.
   b. qué conviene que Juan le pida a Ana.

3. Para adaptar la aplicación a lo que se necesita en la India, se agrega como opción de transporte la manada de elefantes, que la va a programar Ravi en Bangalore. Las condiciones para poder llevar un envío, y lo que se hace cuando se le agrega un envío, no tienen nada que ver con el resto. ¿Qué indicaciones hay que darle a Ravi para que codifique la clase ManadaDeElefantes de forma tal que pueda incluirse en la aplicación? Relacionar con conceptos vistos en la materia, comparar con lo hecho en el punto 1.2.

Punto 2

Abracex S.R.L. exporta naranjas a distintos destinos. Las naranjas se exportan en lotes, de cada lote nos interesa el peso de cada naranja. Cada país impone una condición para aceptar un lote de naranjas, que siempre es "el peso sea mayor a tanto". Ahora, cada país tiene un concepto distinto del peso que le interesa de un lote. P.ej. podría ser que

- para Mozambique la condición es que todas las naranjas pesen al menos 300 gramos, entonces el peso que le interesa es el de la naranja más liviana, y el valor mínimo es 300.
- para Lituania la condición es que el lote pese al menos 10 kg, entonces el peso que le interesa es el total del lote, y el valor mínimo es 10000.
- para Brunei la condición es que la mediana de los pesos de las naranjas sea al menos 250 gramos, entonces el peso que le interesa es la mediana, el valor mínimo es 250.
- para otro país podría ser el concepto "promedio", etc.

Además, para que sea negocio, sólo conviene exportar un lote si tiene al menos 100 naranjas.
Para resolver si un lote es exportable o no a un país, Abracex S.R.L. decide implementar un programa en Haskell, con una función esExportable que recibe como parámetros un lote, el valor mínimo y … una cosa más.
Cada lote se representa como una lista de números, que son los pesos de las naranjas.
La función tiene esta pinta

```
esExportable otro_parámetro valorMinimo lote =
    (length lote > 100) && ((... algo ...) >= valorMinimo)
```

Lo que falta es la condición relacionada con el peso.
Se pide

1. Indicar el parámetro y el código que faltan para reflejar la condición sobre el peso. Indicar qué concepto que vimos en la materia se usó, explicar dónde/cómo se usó, y cuál es la ventaja concreta que nos da en este caso.
2. Comparado con una solución en Prolog, la solución en Haskell que se presenta tiene como ventaja que modelar la condición para cada país es más sencillo.
   Ahora, de poder hacerse en Prolog, habría una ventaja relacionada con las consultas que pueden hacerse a partir del programa, siempre respecto de los destinos posibles de un lote. ¿Cuál sería esta ventaja de armar el programa en Prolog? Relacionar con algún concepto visto en la materia.
3. Ahora Abracex S.R.L. también quiere saber, dada una lista de lotes (o sea, una lista de listas de números), un indicador de cada uno, p.ej. la cantidad, el peso total, etc..
   Eso se resuelve fácil con un map, p.ej.

```
> map length [[3,8,5],[9,12,23,4],[2,4,1,4,32],[2,3]]
[3,4,5,2]
> map sum [[3,8,5],[9,12,23,4],[2,4,1,4,32],[2,3]]
[16,48,43,5]
```

   Ahora quieren saber también el peso de la n-ésima naranja de cada lote, ordenadas de la más pesada a la más liviana. P.ej. el peso de la más pesada, el peso de la 3ra más pesada, el peso de la 7ma más pesada, etc..
   Sería aburridísimo armar funciones distintas para después consultar

```
map masPesada listaDeLotes
map terceraMasPesada listaDeLotes
map septimaMasPesada listaDeLotes
```

   etc.
   Queremos una sola función que resuelva todos estos casos. Se pide
   a. indicar qué parámetro tiene que tener esta función, además del lote
   b. mostrar cómo se usaría en el `map (...) listaDeLotes`, indicando qué concepto asociado al paradigma funcional se está usando.

   No se pide codificar la función.

Punto 3

1. Se tienen estos fragmentos de código, uno en C y otro en Prolog

```
C                       Predicado en PROLOG
a := b + 4;             ... :- ..., A is B + 4, ..(2).., A is C * 2, ..(3)..
// ..(2)..
a := c * 2;
// ..(3)..
```

   El comportamiento no es el mismo en ambos casos. Indicar cuál es la diferencia, relacionándola con conceptos vistos en la materia.
   Pensar en particular en qué pasa con a/A, y con usos de la variable en (2) y (3).

2. ¿Qué diferencia hay entre los predicados p1 y p2 en el programa que sigue? Explicarlo a partir de conceptos relacionados con el paradigma lógico y/o la lógica.

```
p1(X):- condicion1(X), condicion2(X).

p2(X):- condicion1(X).
p2(X):- condicion2(X).
```