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*

negate x = (-1) \* x

f e lista = map (negate . (e+)) ( filter (5 <) lista )

1. ¿Cuáles de los siguientes conceptos se observan en la función f? Indicar todos los lugares donde se están aplicando y qué aportan a la solución.
   1. composición de funciones
   2. expresiones lambda
   3. listas por comprensión
   4. aplicación parcial
   5. orden superior
2. ¿Qué sucede ante las siguientes invocaciones? Justificar.
   1. head (f 1 [1..])
   2. map negate (f 1 [1..])
   3. last (f 1 [1..])
3. ¿Cuál es el problema de definir el tipo de negate de la siguiente manera? Corregirlo.

negate :: Int->Int

*Parte B*

En un sistema para un taller, en el cual se quiere saber cuáles de las mesas que tienen son fuertes, se tiene la siguiente implementación para saber si una mesa efectivamente es fuerte (se asume que el material está correctamente implementado):

#Mesa (V.I.: patas, material)

esFuerte

^ self esDeMaterialResistente and: [self enMalEstado not]

esDeMaterialResistente

^material esResistente

enMalEstado

^ patas = 3

Ahora se le agregan nuevos objetos que representan a mesas de trabajo, que tienen más requisitos para ser consideradas fuertes y trabajar sobre ellas. En particular, se debe validar que la tabla sea lisa. Se propone agregar lo siguiente a la solución:

#MesaDeTrabajo (subclase de Mesa) vi: tabla

esFuerteDeTrabajo

^ (self esDeMaterialResistente and: [self enMalEstado not]) and: [tabla lisa]

(Se asume que hay un método llamado lisa que está definido para las tablas)

1. El desarrollador que planteó la solución justifica: "La solución funciona correctamente y no tuve que tocar nada de lo hecho anteriormente sino sólo agregar código. La herencia fue útil, ya que los objetos de MesaDeTrabajo entienden los métodos esDeMaterialResistente y enMalEstado sin que los vuelva a definir". ¿Es cierto lo que dice? Justificar.
2. Al margen de lo anterior, ¿qué problema/s podés marcar en la solución propuesta? Justificar.
   1. mala delegación
   2. falta polimorfismo
   3. repite lógica
   4. rompe el encapsulamiento
   5. no es expresiva
3. Plantear una mejor solución. Explicar qué otro/s concepto/s son utilizados.

*Parte C*

Dada la siguiente base de conocimientos:

%material(nombre, tipo, grosor)

material(pino, madera, 20).

material(algarrobo, madera, 10).

material(acero, metal, 25).

material(aluminio, metal, 12).

esResistente(algarrobo).

esResistente(Material) :- material(Material, \_, Grosor), Grosor > 15.

Se tienen las siguientes implementaciones de un predicado para saber si todos los materiales de un determinado tipo son resistentes:

|  |  |
| --- | --- |
| todosResistentes(Tipo):-  findall(Material, material(Material, Tipo,\_),  Materiales),  resisten(Materiales).  resisten([Material | Materiales]):-  esResistente(Material),  resisten(Materiales). | todosResistentes(Tipo):-  forall(esResistente(Material),  material(Material, Tipo, \_)). |

1. Ante la consulta:  
   **?- todosResistentes(madera).**   
   Ambas soluciones responderán falso, cuando en realidad todas las maderas son resistentes. Explicar cuál es el problema en cada solución e indicar cómo se soluciona.
2. Si se solucionan los problemas mencionados en el punto anterior, ¿el predicado todosResistentes puede ser usado para conocer algún tipo de material que cumpla lo pedido? Justificar conceptualmente la respuesta.
3. Comparar ambas soluciones en términos de declaratividad.