


Condiciones de aprobación

Para aprobar es necesario simultáneamente: <ul style="list-style-type: none"> • completar el 60% del examen, y • obtener al menos la mitad de los puntos en cada paradigma. 	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

Se cuenta con la siguiente información:

-- Para cada medicamento:

`amoxicilina` = cura "infección"

`bicarbonato` = cura "picazón"

`ibuprofeno` = cura "dolor" . cura "hinchazón"

`todosLosMedicamentos` = [`amoxicilina`, `bicarbonato`, `ibuprofeno`]

`cura sintoma` = filter (/= sintoma)

-- Para cada enfermedad / conjunto de síntomas:

`malMovimiento` = ["dolor"]

`varicela` = repeat "picazón" ¹

-- Asumimos que existe al menos un medicamento capaz de curar cada enfermedad:

`mejorMedicamentoPara` sintomas = find (curaTodosLos sintomas) todosLosMedicamentos

`curaTodosLos` sintomas medicamento = medicamento sintomas == []

Se pide:

1. Definir el tipo de un medicamento genérico, y de `curaTodosLos`.
2. ¿Cuáles son los dos conceptos funcionales más importantes aplicados en `mejorMedicamentoPara` y de qué sirvieron?
3. ¿Qué responde esta consulta? Justificar conceptualmente. En caso de errores o comportamientos inesperados, indicar cuáles son y dónde ocurren.
> `mejorMedicamentoPara` varicela

Si se reemplaza `todosLosMedicamentos` con lo de abajo, ¿qué respondería la consulta anterior? Justificar conceptualmente.

`sugestion` sintomas = []

`todosLosMedicamentos` = [`sugestion`, `amoxicilina`, `bicarbonato`, `ibuprofeno`]

Parte B

Tenemos un predicado `toma/2` que relaciona a una persona con aquella bebida que le gusta tomar. La bebida puede ser cerveza, que tiene una variedad, un amargor y un porcentaje de alcohol (0 si es cerveza sin alcohol), o vino, que tiene un tipo y una cantidad de años de añejamiento. El vino siempre tiene alcohol. Y también existen gaseosas varias.

<pre>toma(juan, coca). toma(juan, vino(malbec, 3)). toma(daiana, cerveza(golden, 18, 0)). toma(gisela, cerveza(ipa, 52, 7)). toma(edu, cerveza(stout, 28, 6)).</pre>	<pre>tieneProblemas(Persona):- findall(C,(toma(Persona, cerveza(C,_,A)), A>0),Cs), findall(V, toma(Persona, vino(V,_)), Vs), findall(T, toma(Persona, T), Ts), length(Cs, CCs), length(Vs, CVs), length(Ts, CTs), CTs is CCs + CVs.</pre>
--	--

1. Justificar V o F

a. No se repite código, dado que la estructura de las bebidas alcohólicas son distintas.

¹ repeat x = x : repeat x

- b. La solución planteada para `tieneProblemas/1` es declarativa.
 - c. La solución planteada no utiliza polimorfismo.
2. Explique y justifique cuál es el significado de lo que se estaría consultando con el siguiente código:
?- `tieneProblemas(P)`.
3. Implemente una solución superadora de `tieneProblemas/1`.

Parte C

Tenemos un pequeño programa que modela superhéroes y villanos de la empresa “Maravilla” de comics (y películas, series, videojuegos, etc.). Nos piden modelar los ataques de los personajes a otros personajes, y para esto debemos representarlos. Los personajes pueden ser héroes o villanos. Un héroe sólo puede atacar a sus enemigos, mientras que un villano puede atacar a cualquier otro personaje. En el caso del villano, si ataca a un enemigo, ese enemigo recibe el doble de daño. Un héroe ataca con tanto poder como la suma de sus poderes potenciados por sus aliados. Cada uno tiene un determinado poder, el cual es multiplicado por la cantidad de aliados del héroe. Los villanos, en cambio, tienen armas, que producen un determinado daño total sobre un área de efecto dada. El poder se calcula como el daño total dividido por el área de efecto.

Se tiene el siguiente modelo base planteado:

Wollok <pre> class Personaje { var enemigos = [] method recibirDaño(cantidad) {...} } class Heroe inherits Personaje { var habilidades = [] var aliados = [] method atacar(personaje) { if (enemigos.contains(personaje)) personaje.recibirDaño(self.poder()) } method poder() = habilidades.sum({h => h.poder() * aliados.size()}) } class Villano inherits Personaje { method atacar(personaje) { personaje.recibirDaño(self.poder()) if (enemigos.contains(personaje)) personaje.recibirDaño(self.poder()) } method poder() = armas.sum({a => a.daño() / a.areaDeEfecto()}) } class Habilidad { var poder } class Arma { var daño var areaDeEfecto }</pre>	Smalltalk <pre> #Personaje (v.i. enemigos) >>recibirDaño: cantidad “Something...” #Heroe (subclase de Personaje - v.i. habilidades, aliados) >>atacar: personaje (enemigos includes: personaje) ifTrue: [personaje recibirDaño: self poder] >>poder ^habilidades sum: [:h h poder * aliados size] #Villano (subclase de Personaje - v.i. armas) >>atacar: personaje personaje recibirDaño: self poder. (enemigos includes: personaje) ifTrue: [personaje recibirDaño: self poder] >>poder ^armas sum: [:a a daño / a areaDeEfecto] #Habilidad (v.i. poder) #Arma (v.i. daño, areaDeEfecto)</pre>
---	--

1. Responder Verdadero o Falso y justificar:
 - a. El ataque de un héroe no maneja correctamente el caso de atacar a un personaje que no es enemigo.
 - b. La lógica de condición de saber si el personaje a atacar es enemigo y la consecuencia de *atacarlo* (es decir, que el otro reciba daño) se repite en ambas clases, por lo que dicha lógica debería estar en la superclase *Personaje*.
 - c. Puede verse el uso polimórfico de villanos y héroes en la solución.
2. Proponer una solución alternativa que corrija las observaciones realizadas en el punto anterior y cualquier otra mejora que sea apropiada.