Condiciones de aprobación

Para aprobar es necesario simultáneamente: • 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

Queremos hacer un pequeño programa para modelar superhéroes y villanos de la empresa "Maravilla". Nos piden implementar los ataques de los personajes a otros personajes teniendo en cuenta que los personajes pueden ser héroes o villanos, y debe ser posible que un héroe se convierta en villano y viceversa. Un héroe sólo puede atacar a sus enemigos, mientras que un villano puede atacar a cualquier personaje. Luego de dañar al personaje atacado, el otro personaje lo considerará su enemigo.

Un héroe ataca con tanto poder como la suma de los poderes de sus habilidades multiplicada por la cantidad de aliados del héroe. Un villano en cambio usa su arma para atacar, la cual produce un determinado daño que se calcula como su daño base dividido por su área de efecto.

Tenemos el siguiente planteo para la solución:

```
class Personaje {
  const enemigos = #{}
  method recibirDaño(cantidad) {
    /* Asumir que funciona */
  }
  method agregarEnemigo(personaje) {
    enemigos.add(personaje)
  }
}
class Villano inherits Personaje {
  var property arma
  method atacar(personaje) {
    personaje.recibirDaño(self.dañoDeArma())
    personaje.agregarEnemigo(self)
  }
  method dañoDeArma() = arma.dañoBase() /
    arma.areaDeEfecto()
}
class Arma {
  var property dañoBase
  var property areaDeEfecto
}
```

```
class Heroe inherits Personaje {
  const habilidades = #{}
  const aliados = #{}

  method atacar(personaje) {
    if (not enemigos.contains(personaje)) {
      return "El personaje atacado
              no es un enemigo"
    }
    personaje.recibirDaño(self.poder())
    personaje.agregarEnemigo(self)
  }

  method poder() = aliados.size() *
    habilidades.sum({h => h.poder()})
}

class Habilidad {
  var property poder
}
```

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. Hay lógica común entre villanos y héroes que podría generalizarse.
   c. No hay problemas de delegación en la solución.
   d. Los héroes son polimórficos con las habilidades, ya que ambos entienden el mensaje `poder()`.
   e. Para que un héroe se convierta en villano alcanza con instanciar Villano, inicializándolo con el arma que corresponda y los enemigos que tenía la instancia original de la clase Heroe.
2. Proponer una solución alternativa realizando las mejoras que consideres apropiadas en base a las observaciones del punto anterior.

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.

```
toma(juan, coca).
toma(juan, vino(malbec, 3)).
toma(daiana, cerveza(golden, 18, 0)).
toma(gisela, cerveza(ipa, 52, 7)).
toma(gisela, vino(malbec, 3)).
toma(edu, cerveza(stout, 28, 6)).
```

```
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.
```

1. Explicar cuál es el objetivo del predicado `tieneProblemas/1` (no explicar la forma en que lo resuelve, sólo para qué sirve) y cuál será la respuesta a la consulta:
   ?- tieneProblemas(juan).
2. Responder Verdadero o Falso y **justificar**:
   a. El predicado `tieneProblemas/1` no es inversible.
   b. La solución planteada para `tieneProblemas/1` es declarativa.
   c. La solución planteada podría mejorarse usando polimorfismo.
3. Implementar una solución de `tieneProblemas/1` con las mejoras que consideres apropiadas.

Parte C

Se cuenta con la siguiente información:

```
-- Para cada medicamento:
amoxicilina = cura "infección"
bicarbonato = cura "picazón"
sugestion _ = []

cura sintoma = filter (/= sintoma)

-- Para cada enfermedad / conjunto de síntomas:
malMovimiento = ["dolor agudo", "hinchazón"]
varicela = repeat "picazón" [1]

mejorMedicamentoPara sintomas = head . filter (idealPara sintomas)
idealPara sintomas medicamento = medicamento sintomas == []
```

Se pide:

1. Definir el tipo `Medicamento` en base al modelo dado, y explicitar el tipo de la función `mejorMedicamentoPara`.
2. Explicar qué beneficio aporta el uso de orden superior en la definición de `mejorMedicamentoPara`.
3. Definir y explicitar el tipo del **`ibuprofeno`**, para que pueda usarse como medicamento, que cure tanto el `"dolor agudo"` como la `"hinchazón"` si es de más de 500 miligramos, y de lo contrario cure el `"dolor moderado"`. Armar una lista de medicamentos que incluya amoxicilina e ibuprofeno de 400 miligramos. En caso de que se esté aprovechando algún concepto importante para llevarlo a cabo, mencionarlo.
4. ¿Qué sucederá al evaluar las siguientes consultas? Justificar conceptualmente. En caso de errores o comportamientos inesperados, indicar cuáles son y dónde ocurren.
   a. `mejorMedicamentoPara malMovimiento (repeat bicarbonato)`
   b. `mejorMedicamentoPara varicela [sugestion, bicarbonato, amoxicilina]`

[1] `repeat x = x : repeat x`