Paradigmas de Programación

Examen Final

15/02/2020

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 ayudar a definir el resultado de una serie de rondas del juego piedra, papel o tijera entre varias personas que juegan al mismo tiempo. Sabemos que se gana una ronda cuando se elige una opción que le gana a lo que eligieron los demás jugadores. Tenemos la siguiente implementación en Prolog:

```
eligio(1, juan, piedra).
eligio(2, juan, piedra).
eligio(3, juan, tijera).

eligio(1, ana, papel).
eligio(2, ana, piedra).
eligio(3, ana, tijera).
```

```
eligio(1, tito, tijera).
eligio(2, tito, papel).
eligio(3, tito, piedra).

leGana(piedra, tijera).
leGana(tijera, papel).
leGana(papel, piedra).
```

```
ganoRonda(Ronda, Jugador):-
  eligio(Ronda, Jugador, Gana),
  eligio(Ronda, _, Pierde),
  leGana(Gana, Pierde).
```

1. La solución propuesta para el predicado ganoRonda/2 no cumple con la consigna. Explicar el motivo, comparando lo que se buscaba con lo que hace realmente, y demostrar esa diferencia de funcionalidad con un ejemplo a partir de la información de base de conocimientos dada.
2. Proponer una nueva solución para ganoRonda/2 que cumpla con lo pedido.
3. En base a la nueva solución, ¿cómo se podría hacer para saber quiénes ganaron alguna ronda? ¿Y en qué rondas no ganó nadie? Explicar qué características del paradigma ayudan a resolver estos problemas.

Parte B

Dado el siguiente dominio:

```
data Persona = Persona { nombre :: String, edad :: Int, bebidas :: [Bebida] }
type Bebida = Persona -> Persona

cafe (Persona n e bs) = Persona n (e + 1) (cafe:bs)
cindor (Persona n e bs) = Persona ("Energizado " ++ n) e (cindor:bs)
gatorade (Persona n e bs) = Persona n e (gatorade:bs)

tomar persona unaBebida = unaBebida persona
puedeTomar persona unaBebida = ((< 60) . edad . tomar unaBebida) persona
juan = Persona "Juan" 23 []
```

1. ¿Qué pasará si se hacen las siguientes consultas? Justificar conceptualmente.
   ```
   > puedeTomar cafe juan
   > edad juan
   ```
2. ¿Hay lógica repetida entre las bebidas? En caso positivo, proponer una solución para evitarlo e indicar qué conceptos se aprovechan. En caso negativo, justificar por qué no.
3. Dada la siguiente función
   ```
   f persona = head . map (tomar persona) . filter (puedeTomar persona)
   ```
   a. Indicar el tipo de la función f.

b. Proponer un nombre más expresivo para la función.
c. Si tenemos la siguiente lista infinita de bebidas:
`muchasBebidas = cycle [cafe, cindor, gatorade]` --[1]
¿qué sucede al evaluar f para juan y muchasBebidas? En caso que termine, indicar cómo se reduce a su valor final. En caso que no termine, justificar por qué y dar un contraejemplo.

Parte C

En una agencia de publicidad gráficas existen dos tipos de publicidades:

- Publicidades tradicionales: se publican en diarios y revistas de la agencia, y tienen un costo para el cliente que varía según el plan que contrate para esa publicidad, permitiendo cambiarlo más adelante:
  - Estandar: $1000.
  - Platinum: $800 si el cliente tiene activas más de 3 publicidades. Si no, $1100.
  - Gold: $1200 / cantidad de publicidades activas del cliente.
- Publicidades no tradicionales: se publican en distintos medios a elección, cada uno de los cuales tiene un costo propio. El costo de la publicidad no tradicional es la suma de estos costos.

Las publicidades que no están activas no tienen costo. En cualquier momento podrían dejar de estar activas a pedido del cliente.

```
class Cliente {
  var property publicidades = []
}
class PublicidadNoTradicional {
  var property medios = []
  var property estaActiva = true
  method costo() {
    if(estaActiva)
      return medios.sum {medio =>
        medio.costo()}
    else return 0
}
class PublicidadTradicional {
 var property plan
 var property estaActiva = true
 var cantPublicidadesActivas = 0

  method costo(cliente) {
   cantPublicidadesActivas = cliente.publicidades()
     .count({publicidad =>
publicidad.estaActiva()})

   if(not estaActiva) return 0

   if(plan == "estandar") return 1000
   if(plan == "platinum") {
    if(3 < cantPublicidadesActivas) return 800
    else return 1100
   }
   if(plan == "gold")
     return 1200 / cantPublicidadesActivas
   return "Plan inválido"
  }
}
```

1. Responder V/F, **justificando** en todos los casos:
   a. No hay lógica repetida en la solución.
   b. Hay problemas en el manejo del estado del sistema.
   c. Si se quisiera definir un método costoTotal en la clase Cliente, se podría resolver fácil gracias a que todas las publicidades entienden el mensaje costo.
   d. Hay problemas de delegación en la clase PublicidadTradicional.
   e. Retornar un string si se configuró mal el plan no es conveniente. Sería mejor retornar -1.
2. Desarrollar una solución superadora para el problema. En caso de aprovechar conceptos del paradigma que no se estuvieran aprovechando en la solución original, explicarlo brevemente.

[1] La función `cycle` genera una lista infinita repitiendo los elementos de la que recibe por parámetro:
`cycle [1, 2, 3] --> [1, 2, 3, 1, 2, 3, 1, 2 ... ]`