Paradigmas de Programación | Examen Final | 21/12/2024

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

Se desea modelar los tipos de cuentas que pueden contratar los usuarios de una plataforma de streaming. Existen tres tipos de cuentas:

- Cuenta Básica: Puede reproducir hasta un límite de horas de contenido al mes, luego no se permite.
- Cuenta Premium: No tiene límite de reproducción. Además, puede descargar contenido con un límite de GB.
- Cuenta Familiar: Es como una cuenta Premium pero permite múltiples perfiles de usuarios, cada uno con límites distintos.

Además, se desea:

- Obtener un string que muestre el tipo de cuenta, el nombre del usuario principal, y la información específica de cada tipo de cuenta: horas restantes (en una cuenta básica), espacio disponible (en cuentas premium o familiar), y perfiles activos (en cuentas familiares).
- Todas las cuentas deben poder **cambiar de tipo** en cualquier momento, conservando sus datos relevantes.

Contamos con la siguiente solución:

```
class Cuenta {
  var property tipo
  var property usuarioPrincipal
  var property horasConsumidas
  var property limiteHoras
  var property espacioConsumido
  var property limiteEspacio
  var property perfiles

  method estado() {
    if (tipo == "Básica") {
      return "Tipo: " + tipo + ", Usuario: " +
         usuarioPrincipal +", Horas restantes: " +
         (limiteHoras - horasConsumidas)
    } else if (tipo == "Premium") {
      return "Tipo: " + tipo + ", Usuario: " +
         usuarioPrincipal + ", Espacio disponible: " +
         (limiteEspacio - espacioConsumido) + "GB"
    } else {
      return "Tipo: " + tipo + ", Usuario: " +
         usuarioPrincipal + ", Perfiles: " +
         perfiles.size()
    }
  }
```

```
// (sigue de la class Cuenta)
 method reproducirContenido(horas) {
    if (tipo == "Básica") {
      if (horasConsumidas + horas > limiteHoras) {
        return "No podés reproducir más contenido
           este mes."
      }
      horasConsumidas += horas
    } else {
      return "Contenido reproducido."
    }
  }

  method descargarContenido(gb) {
    if (tipo == "Premium" || tipo == "Familiar") {
      if (espacioConsumido + gb > limiteEspacio) {
        return "No hay suficiente espacio."
      }
      espacioConsumido += gb
    } else {
      return "Tu plan no permite descargas."
    }
  }
}
```

1. **Responder Verdadero o Falso y justificar:**
   a. Sin modificar los métodos existentes, puede agregarse fácilmente un nuevo tipo de cuenta, por ejemplo, una cuenta "Pro" que permite descargas ilimitadas pero no reproducción sin conexión.

   b. Si se agrega un método `transferirContenido(cuenta1, cuenta2)`, que permita transferir contenido descargado entre cuentas premium o familiares, pero no entre cuentas básicas:

```
method transferirContenido(cuenta1, cuenta2) {
  cuenta1.descargarContenido(-5) // Libera 5GB en cuenta1
  cuenta2.descargarContenido(5)  // Consume 5GB en cuenta2
}
```

   Si una de las cuentas no tiene suficiente espacio disponible, no se afecta el estado de la otra.

2. **Proponer una solución mejorada:** Diseñar una solución que respete las buenas prácticas de orientación a objetos, evite repeticiones de lógica y facilite la extensión a nuevos tipos de cuentas sin modificar código existente.

Parte B

Queremos estudiar el comportamiento de la gente al elegir alimentos. Contamos con:

```
data Alimento = Comida {
    nombre :: [Char],
    calorias :: Float,
    nutrientes :: [String]
}
ana = [esBajoEnCalorias, tieneProteinas]
```

```
esBajoEnCalorias :: Alimento -> Bool
esBajoEnCalorias comida = calorias comida < 100

tieneProteinas :: Alimento -> Bool
tieneProteinas = elem "proteinas" . nutrientes
```

1. Definir el tipo de ana.

2. Queremos determinar si un alimento tiene un nutriente en particular, no necesariamente proteínas:
   a. Definir la función `tieneNutriente` y explicitar su tipo.
   b. Volver a definir ana usando `tieneNutriente` en vez de `tieneProteinas`, indicando qué concepto del paradigma funcional se aprovecha en esta solución que no se usaba en la definición anterior de ana.

3. Se cuenta con una función que permite establecer, dada una lista de alimentos, cuáles elegiría una persona según sus requisitos (por ejemplo, ana), asumiendo que elige un alimento que cumpla al menos 1.

```
alimentosElegidos :: [Alimento -> Bool] -> [Alimento] -> [Alimento]
alimentosElegidos requisitos alimentos = filter (f requisitos) alimentos
f :: [Alimento -> Bool] -> Alimento -> Bool
f requisitos alimento = any (\requisito -> requisito alimento) requisitos
```

   Responder verdadero o falso, justificar y corregir la implementación en caso de ser necesario:
   a. La solución funciona correctamente.
   b. La solución es poco declarativa.
   c. La solución es poco expresiva.
   d. Asumiendo que funciona como se explica, si se la invoca con una lista de alimentos infinita, no se podrá obtener ninguna respuesta independientemente de cuáles sean los requisitos de la persona.

Parte C

Dada la siguiente base de conocimiento:

```
%esAutorDe(Autor, Obra).
esAutorDe(tolkien, elSeniorDeLosAnillos).
esAutorDe(tolkien, elHobbit).
esAutorDe(rowling, harryPotterYLaCamaraSecreta).
esAutorDe(rowling, unaVacanteImprevista).
esAutorDe(miguelCervantes, donQuijote).
esAutorDe(borges, elAleph).
```

```
%genero(Obra, Genero).
genero(elSeniorDeLosAnillos, fantasia).
genero(elHobbit, fantasia).
genero(harryPotterYLaCamaraSecreta, fantasia).
genero(unaVacanteImprevista, novela).
genero(donQuijote, novela).
genero(elAleph, cuento).
```

Queremos identificar si un autor se dedica exclusivamente a un único género literario. Por ejemplo, **tolkien** escribe únicamente fantasía, mientras que **rowling** no, ya que ha escrito tanto fantasía como novelas. Una consultora propuso la siguiente solución:

```
autorEspecifico(Autor) :-
    esAutorDe(Autor, _),
    forall(genero(Obra, _), esAutorDe(Autor, Obra)).
```

1. Explicar por qué esta solución no cumple con el problema planteado. Usar lenguaje coloquial para describir qué hace realmente la solución propuesta.

2. Proponer una alternativa que funcione correctamente para resolver consultas como:
   - `autorEspecifico(tolkien)`
   - `autorEspecifico(Quien)` (y que sea inversible)

3. Indicar en qué partes de su solución aparecen los conceptos de:
   - **Orden Superior**
   - **Inversibilidad**