Paradigmas de Programación **Examen Final** **17/02/2024**

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

Dadas las siguientes definiciones

`habilidades :: Persona -> [Habilidad]`	`sirve :: Problema -> Habilidad -> Bool`

1. Usando las funciones `habilidades` y `sirve`, definir una función que dado un problema y una lista de personas permita saber qué personas tienen alguna habilidad que sirva para el problema recibido.

```
podrianAyudar problema personas = …
```

2. Indicar qué concepto/s de los siguientes se está/n utilizando en la solución anterior y para qué:
   - Orden Superior
   - Composición
   - Aplicación Parcial
   - Pattern matching

3. Si una persona tuviera una lista infinita de habilidades, ¿cómo se comportaría `podrianAyudar` si dicha persona estuviera en la lista recibida?

Parte B

Asumiendo que en la base de conocimiento hay numerosos hechos de los siguientes predicados:

```
tiene(Persona, Cosa).
vale(Cosa, Valor).
```

1. Completar lo necesario para que `todoLoQueTieneEsMasValioso` permita saber si todo lo que tiene la primera persona es más valioso que lo que tiene la segunda.

```
todoLoQueTieneEsMasValioso( Persona1, Persona2 ):-

    forall((.....), ValorCosaValiosa > OtroValor ).
```

2. ¿Es posible usar `todoLoQueTieneEsMasValioso` para consultar si nadie cumple con tener todas cosas más valiosas que una persona indicada?
   a. Si es posible, explicar qué es lo que permite dicho uso.
      En caso de no ser posible con la solución del punto 1 tal y como está, explicar por qué, y hacer los cambios necesarios de modo que sí lo permita.
   b. Mostrar qué consulta debería realizarse para saber si nadie tiene todas cosas más valiosas que pedro.

Parte C

En nuestro sistema, para que una película rompa estereotipos, en principio su protagonista no debe ser varon. Además se debe cumplir: en el caso de las películas de aventura, que ese protagonista no sea rescatado; en el caso de las películas de terror, que todos los personajes sean rescatados; y en el caso de las de comedia, que sólo tengan un personaje. Se sabe que los personajes animados no tienen género definido y nunca son rescatados, y los personajes actuados pueden tener distintos géneros y ser o no rescatados en la película. Se tiene la siguiente solución:

```
class Pelicula {
  var personajes = []
  method protagonista() = personajes.first()
}
class PeliAventura inherits Pelicula {
  method rompeEstereotipos(){
    return self.protagonista().sosActuado() and self.protagonista().genero() != "varon"
      and self.protagonista().esRescatado().negate() }
}
class PeliTerror inherits Pelicula {
  method rompeEstereotipos(){
    return self.protagonista().sosActuado() and self.protagonista().genero() != "varon"
      and personajes.all({p => p.esRescatado()})  }
}
class PeliComedia inherits Pelicula {
  method rompeEstereotipos(){
    return self.protagonista().sosActuado() and self.protagonista().genero() != "varon"
      and personajes.size() == 1  }
}
class PersonajeAnimado {
  method sosActuado() = false
  method esRescatado() = false
}
class PersonajeActuado {
  var genero
  var esRescatado
  method sosActuado() = true
  method genero() = genero
  method esRescatado() = esRescatado
}
```

1. Responder verdadero o falso y **justificar conceptualmente** en todos los casos:
   a. Escribiendo un método `elProtagonistaNoEsVaron` en la superclase y usándolo en las subclases se elimina toda repetición de lógica.
   b. Se está rompiendo el encapsulamiento de los personajes.
   c. Como no hay ifs, se está haciendo un buen uso del polimorfismo.
2. Codificar una nueva solución arreglando lo que haga falta en base a las respuestas anteriores. Incluir un diagrama estático de la nueva solución.