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.	estas incompletas. 

Parte A

Tenemos el siguiente código para monitorear cómo el granjero alimenta a sus caballos y chanchos. Sabemos que a los caballos los alimenta diferente que a los chanchos, y a su vez, que no a todos los chanchos los alimenta igual: a los chanchos gordos les da menos alimento para que no engorden tanto.

Wollok

```
class Granjero {
  const animales = []
  method alimentarAnimales(){
  self.caballos().forEach({ caballo =>
      caballo.energizar(caballo.racionDiaria())
   })
   self.chanchos().forEach({ chancho =>
     if(chancho.kilos() > 50)
       chancho.aumentarKilos(2)
     else
       chancho.aumentarKilos(3)
   })
  }

  method caballos() =
   animales.filter({animal=>animal.esCaballo()})

  method chanchos() =
   animales.filter({animal=>animal.esChancho()})
}

class Caballo {
  var racionDiaria
  var energia
  method racionDiaria() = racionDiaria
  method esChancho() = false
  method esCaballo() = true
  method energizar(cant){
     energia *= 1+cant/100}
}

class Chancho{
  var kilos
  method kilos() = kilos
  method esChancho() = true
  method esCaballo() = false
  method aumentarKilos(cant){kilos += cant}
}
```

Smalltalk

```
#Granjero (v.i. animales)
>> alimentarAnimales
  self caballos do: [:caballo |
    caballo aumentarEnergia: caballo
racionDiaria
  ].
  self chanchos do: [:chancho |
    chancho kilos > 50 ifTrue: [
      chancho aumentarKilos: 2
    ] ifFalse: [
      chancho aumentarKilos: 3
    ]
  ].

>> caballos
  ^ animales select: [:animal |
      animal esCaballo
    ]

>> chanchos
  ^ animales select: [:animal |
      animal esChancho
    ]

#Caballo (v.i. energia racionDiaria)
>> esChancho
  ^ false
>> esCaballo
  ^ true
>> energizar: cant
  energia := energia * (1+(cant/100))
>> racionDiaria
  ^ racionDiaria

#Chancho (v.i. kilos)
>> kilos
  ^ kilos
>> esChancho
  ^ true
>> esCaballo
  ^ false
>> aumentarKilos: cant
  kilos := kilos + cant
```

1. Indicar verdadero o falso y **justificar** en cada caso.
    a. Para que la solución propuesta funcione no es necesario que exista una superclase Animal de la cual hereden Caballo y Chancho.
    b. La solución asigna bien las responsabilidades. Es correcto que sea el granjero quien determine cómo deben alimentarse chanchos y caballos, ya que se asemeja más a la realidad.
    c. La solución hace buen uso de polimorfismo, ya que tanto chanchos como caballos entienden los mensajes esChancho y esCaballo.
    d. Es posible adaptar la solución para que, de querer agregar gallinas al modelo a las cuales también hay que alimentar a su manera, no haya que cambiar la clase Granjero, ni la clase Caballo ni la clase Chancho.
2. Desarrollar una nueva solución que mejore los aspectos negativos detectados.

Parte B

Luego de leer el enunciado de Objetos, un desarrollador generó esta solución en Prolog, para poder saber como quedan los animales luego de que el granjero los alimenta.

```
% relaciona granjero con animales que pueden ser:
% chancho(kilos) o caballo(energia, ración)
granjero(beto, chancho(24)).
granjero(beto, caballo(150,20)).
granjero(beto, chancho(120)).
granjero(julieta, caballo(120,10)).

alimentar(Granjero, Animal):-
    granjero(Granjero, caballo(Energia, Ración)),
    Energia = Energia * (1 + Racion/100).
```

```
alimentar(Granjero, Animal):-
    granjero(Granjero, chancho(Kilos)),
    Kilos =< 50,
    Kilos = Kilos + 3.

alimentar(Granjero, Animal):-
    granjero(Granjero,chancho(Kilos)),
    Kilos = Kilos + 2.
```

1. La solución tiene algunos problemas conceptuales. Indicar los que considere más importantes (al menos tres) y justificar por qué son errores.
2. Implemente una solución alternativa que arregle los problemas detectados, teniendo en cuenta que debe respetar la consigna original.
3. ¿Es inversible la nueva solución? Ejemplificar mostrando cómo responde a consultas diversas.

Parte C

Dada la siguiente función en Haskell:

```
f  h  p = any (<h). map p . snd
```

1. Dar el tipo de la función f.
2. Mejorar la función f en términos de expresividad.
3. Dar dos ejemplos de invocación, uno que termine y otro que no, que sirvan para explicar la estrategia de evaluación de Haskell.