FINAL DE PARADIGMAS DE PROGRAMACION 23/09/2010

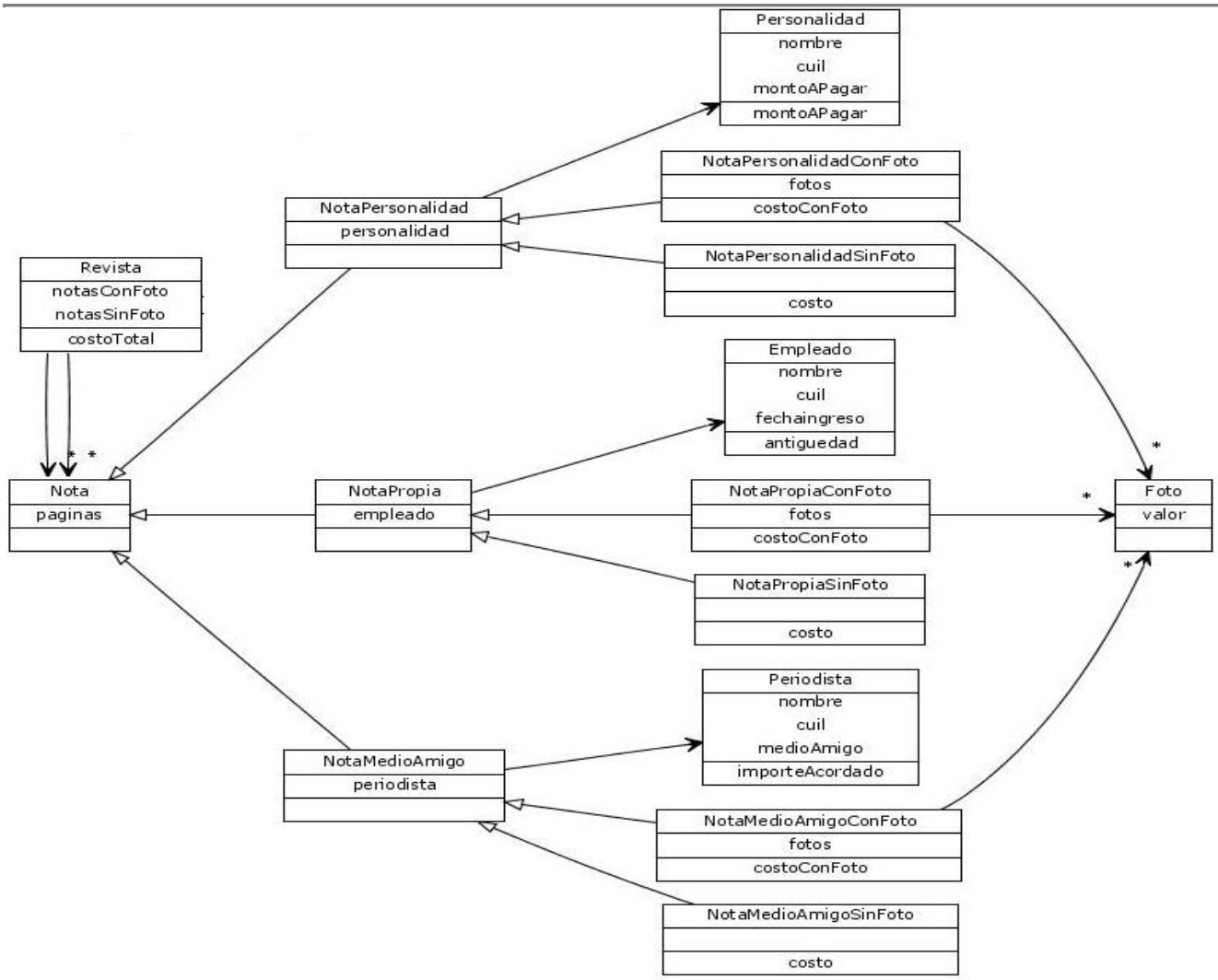
Punto 1

El director de una popular revista de circulación nacional contrató a un especialista en sistemas de una desconocida universidad y le contó su situación:

> *Se quiere saber el costo total de las notas que publica la revista.*
> *El costo de cada nota se calcula según su tamaño (la cantidad de páginas) y su autor.*
> - *A las notas hechas por empleados propios se les paga $100 + 2$ por cada año de antigüedad por cada página.*
> - *La revista tiene acuerdos con otros medios amigos, por los que cuando publica una nota hecha por algún periodistas de dichos medios, se les paga por página el importe acordado (Actualmente, para los periodistas de La Gaceta son $110; Noticias Argentinas $130, La nueva $120, pero los valores podrían cambiar en un futuro y podrían sumarse nuevos medios).*
> - *Cuando la revista publica notas que pide especialmente a diversas personalidades, acuerda con cada una un monto por página.*
>
> *Independientemente de quien sea el autor de la nota, hay algunas que incluyen fotografíasa y otras que son simplemente texto. A las que tienen fotos se le debe sumar el valor de cada una.*

El especialista implementó la siguiente solución:

```
Revista >> costoTotal
  ^(notasSinFoto inject: 0 into: [:tot :nota | tot + nota costo]) +
      (notasConFoto inject: 0 into: [:tot :nota | tot + nota costoConFoto])

NotaPropiaSinFoto >> costo
  ^paginas * (100 + super empleado antiguedad * 2)

NotaPropiaConFoto >> costoConFoto
  ^paginas * (100 + super empleado antiguedad * 2)
        + (fotos inject: 0 into: [:tot :foto | tot + foto valor])

NotaMedioAmigoSinFoto >> costo
  ^paginas * periodista importeAcordado

NotaMedioAmigoConFoto >> costoConFoto
  ^ paginas * periodista importeAcordado + (fotos inject: 0
        into: [:tot :foto | tot + foto valor])

NotaPersonalidadSinFoto >> costo
  ^paginas * personalidad montoAPagar

NotaPersonalidadConFoto >> costoConFoto
  ^ paginas * personalidad montoAPagar + (fotos inject: 0
        into: [:tot :foto | tot + foto valor])

Empleado >> antiguedad
  ^Date today yearsSince: fechaDeIngreso

Periodista >> importeAcordado
  medioAmigo = 'La Gaceta' ifTrue: [^110].
  medioAmigo = 'NoticiasArgentinas' ifTrue: [^130].
  medioAmigo = 'La Nueva' ifTrue: [^110].
```



A la solución la justificó conceptualmente de la siguiente manera:

a. Hay un buen uso del polimorfismo, porque desde **Revista** a todas las **notasConFoto**, sin importar que sean notas de medios amigos, empleados propio o personalidades, les envía el mensaje **costoConFoto** y a todas las notas de la otra colección les envía el mensaje **costo**. De esta manera, que las notas con foto y las notas sin foto estén en dos colecciones separadas simplifica notablemente la solución.
b. Está delegando al máximo, por ejemplo al enviar en **NotaPropiaSinFoto y NotaPropiaConFoto** el mensaje **empleado antigüedad** y que el empleado se ocupe.
c. Es flexible a los cambios, ya que si la revista hace acuerdo con un nuevo medio, sólo hay que modificar el código del método **importeAcordado** y el resto queda como está.
d. La herencia fue muy útil. Por ejemplo la clase abstracta **notaPropia** permite agrupar todo lo común entre **notaPropiaConFoto** y **notaPropiaSinFoto**. Haber definido tres clases fue la mejor forma de evitar repetir código. Lo mismo se utilizó para las notas de medios amigos y las notas de personalidades.
e. El uso de **super** en **notaPropiaConFoto** y **notaPropiaSinFoto** fue fundamental para poder reutilizar el método empleado.
f. Todo esto demuestra que objetos es un paradigma que propicia un desarrollo de software sencillo, genérico y fácilmente modificable.

Desde su experiencia como estudiante de la UTN, realizar lo siguiente:

FINAL DE PARADIGMAS DE PROGRAMACION 23/09/2010

1. Responder punto por punto a las explicaciones planteadas, avalando o refutando con argumentos conceptuales y ejemplos.

2. En base a lo anterior, realizar los cambios en la solución que considere necesarios.

3. Justificar nuevamente el uso de:

   - Polimorfismo
   - Delegación
   - Herencia

4. ¿Fue de utilidad la redefinición y el uso de super?

Punto 2

Se cuenta con las siguientes definiciones en los lenguajes Haskell y Prolog, que funcionan correctamente.

```
concatenar [] ys = ys
concatenar (x:xs) ys = x:concatenar xs ys
```

```
concatenacion([], Ys, Ys).
concatenacion([X|Xs], Ys, [X|Zs]):-concatenacion(Xs, Ys, Zs)
```

1. Explicar en ambas soluciones la aplicación del concepto de pattern matching y unificación.
2. ¿De qué tipos de datos pueden ser los elementos de las listas en cada solución? Explicar similitudes y diferencias entre ambas. En la solución de funcional, explicitar el domino e imagen de la función de la forma más genérica posible.
3. ¿Qué consultas/invocaciones se pueden hacer con cada solución? Mostrar al menos un ejemplo donde se vea un uso similar y al menos 2 donde se vean diferencias. Justificar conceptualmente.

PUNTO 3

Se quiere obtener el primer número primo a partir de un número dado. Para ello se elaboraron dos soluciones que funcionan correctamente. En ambas se asume la existencia de una función llamada primo que dice si un número es primo o no.

Paradigma funcional (Haskell):

```
primerPrimo n = head (filter primo [n, n + 1, ..])
```

Paradigma imperativo (seudo Pascal):

```
function primerPrimo(n: Integer): Integer
begin
      while not(primo(n)) do
            begin
                  n:= n + 1
            end
      primerPrimo := n
end
```

Realizar un análisis comparativo a partir de los siguientes conceptos:

1. Declaratividad
2. Asignación destructiva
3. ¿Qué evita en cada solución que se hagan más cálculos de los necesarios?