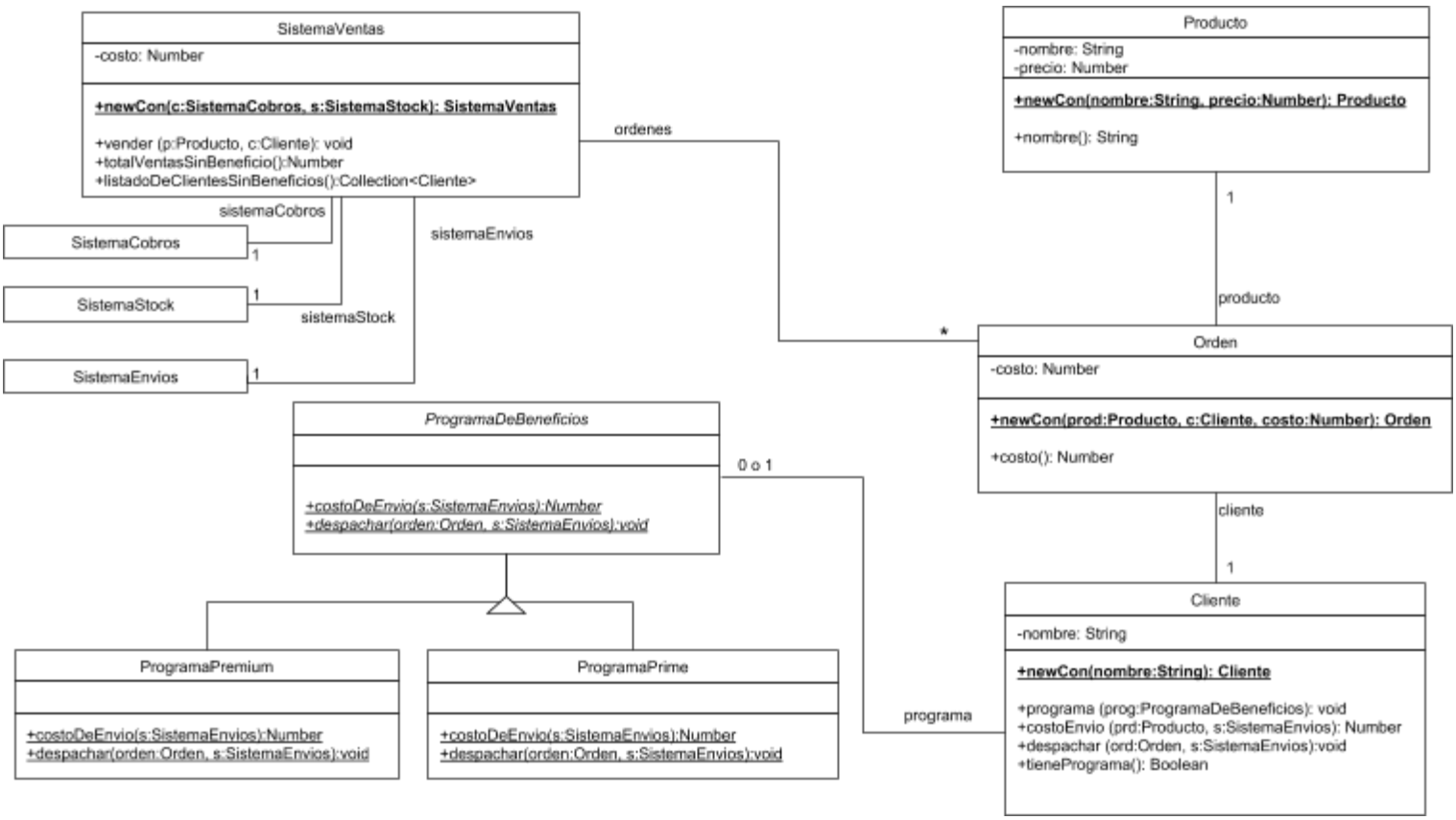
UNLP - Programación Orientada a Objetos 1 2018 - Redictado de 2017
Examen Parcial - Segunda Fecha - 19 de junio de 2018
Solución “en papel” diseñando Cliente sin Programa con nil

Diagrama de Clases



Class Cliente

v.i: ‘ nombre programa ’

“En esta solucion se basa en dejar en nil el programa para diferenciar el comportamiento del cliente cuando no esta vinculado a ningún programa de beneficios. La desventaja de hacer esto es que debemos chequear por nil antes de realizar cualquier acción, y comportamiento que podríamos tener encapuslado en un programa ahora queda en el mismo Cliente. Para ver la diferencia analizar la solucion publicada sin nil.

En esta solucion el sistemaDeEnvios se envía como parametro donde hace falta, esto es indistinto de manejar con nil el programa o no. Es una cuestión de diseño separada, para ver como queda escrito el código de una forma y otra, comparar con la solucion publicada sin nil”

```
class #newConNombre: aString
   ^ self new nombre: aString

#initialize

#nombre: aString
  nombre := aString.
“Notar que programa queda intencionalmente en nil”
  ^self
```

UNLP - Programación Orientada a Objetos 1 2018 - Redictado de 2017
Examen Parcial - Segunda Fecha - 19 de junio de 2018
Solución “en papel” diseñando Cliente sin Programa con nil

```
#programa: unPrograma
   Programa := unPrograma.

#tienePrograma
   ^ programa != nil .

#costoEnvio: unProducto sistemaEnvios: unSistemaDeEnvios
   self tienePrograma ifFalse:[
      ^ unSistemaDeEnvios cotizarEnvio: unProducto]
   ifTrue: [ ^ programa
                  costoEnvio: unProducto sistemaEnvios: unSistemaDeEnvios]

#despachar: unaOrden sistemaEnvios: unSistemaDeEnvios
   self tienePrograma ifFalse:[
      ^ unSistemaDeEnvios despacharEnvioNormal: unaOrden]
   ifTrue: [^ programa
                   despachar: unaOrden sistemaEnvios: unSistemaDeEnvios]
```

Class Producto

```
v.i: ‘ nombre precio’

class #newConNombre: aString precio: aNumber
   ^ self new initConNombre: aString precio: aNumber

#initConNombre: aString precio: aNumber
  nombre := aString.
  precio := aNumber.
  ^self

#nombre
   ^nombre
```

Class Orden

```
v.i: ‘ producto cliente costo’

class #newConProducto: unProducto cliente: unCliente
       costo: aNumber
   ^ self new initConProducto: unProducto cliente: unCliente costo: aNumber

#initConProducto: unProducto cliente: unCliente costo: aNumber
  producto := unProducto.
  cliente := unCliente.
  costo := aNumber.
  ^self

#costo
  ^costo
```

UNLP - Programación Orientada a Objetos 1 2018 - Redictado de 2017
Examen Parcial - Segunda Fecha - 19 de junio de 2018
Solución “en papel” diseñando Cliente sin Programa con nil

Class abstract ProgramaDeBeneficios

v.i: ‘’

```
abstract #costoDeEnvio: unProducto sistemaEnvios: unSistemaDeEnvios
abstract #despachar: unaOrden
```

Class ProgramaDeBeneficios subclass: ProgramaPrime

v.i: ‘’

```
#descuentoPrime
   ^0.5

#costoDeEnvio: unProducto sistemaEnvios: unSistemaDeEnvios
   ^ (unSistemaDeEnvios cotizarEnvio: unProducto) * self descuentoPrime

#despachar: unaOrden sistemaEnvios: unSistemaDeEnvios
   ^ unSistemaDeEnvios despacharEnvioUrgente: unaOrden
```

Class ProgramaDeBeneficios subclass: ProgramaPremium

v.i: ‘’

```
#costoDeEnvio: unProducto
   ^ 0

#despachar: unaOrden sistemaEnvios: unSistemaDeEnvios
   ^ unSistemaDeEnvios despacharEnvioNormal: unaOrden
```

Class SistemaVentas

v.i: ‘sistemaDeCobros sistemaDeStock sistemaDeEnvios ordenes’

```
class #newConCobros: unSistemaCobros stock: unSistemaDeStock envios: unSistemaDeEnvios
   ^ self new initConCobros: unSistemaCobros stock: unSistemaDeStock envios: unSistemaDeEnvios

#initConCobros: unSistemaCobros stock: unSistemaDeStock envios: unSistemaDeEnvios

  sistemaCobros := unSistemaCobros.
  sistemaDeStock := unSistemaDeStock.
  sistemaDeEnvios := unSistemaDeEnvios.
  ordenes := OrderedCollection new.
  ^self
```

UNLP - Programación Orientada a Objetos 1 2018 - Redictado de 2017
Examen Parcial - Segunda Fecha - 19 de junio de 2018
Solución “en papel” diseñando Cliente sin Programa con nil

```
#vender: unProducto a: unCliente
|costoTotal orden|
   (sistemaDeStock hayStock: unProducto)
    ifTrue:[
       sistemaStock decrementarStock: unProducto.
       costoTotal := unProducto precio +
                     unCliente costoEnvio: unProducto
                               sistemaEnvios: sistemaEnvios.

       orden := Orden newConProducto: unProducto
                      cliente: unCliente
                      costo: costoTotal.
       sistemaDeCobros cobrar: orden.
       unCliente despachar: orden sistemaEnvios: sistemaEnvios.
       ordenes add: orden.
       ^orden
     ]
```

```
#totalVentasSinBeneficios
"Retorna la suma de todas las ordenes hechas a clientes sin programa de beneficios  "
^ (ordenes select:[:orden| orden cliente tienePrograma not])
              sum:[:orden| orden costo]
```

```
#listadoDeClientesSinBeneficios
"Retorna una lista sin repetición de los clientes que no están vinculados a ningún programa, ordenados
por nombre de manera ascendente"

 ( (ordenes collect:[:orden | orden cliente] asSet)
          select:[:cliente | cliente tienePrograma not] )
              asSortedCollection:[:c1 :c2 | c1 nombre > c2 nombre]
```

Playground

```
|sistemaCobros sistemaDeStock sistemaEnvios ventas cliente producto1 producto2 |

ventas := SistemaDeVentas newConCobros: sistemaCobros
                          stock: sistemaDeStock
                          envios: sistemaEnvios.

cliente := Cliente newConNombre: 'Juan'.
producto1 := Producto newConNombre: 'X' precio: 10.
ventas vender: producto1 a: cliente.

cliente programa: Prime new.
producto2 := Producto newConNombre: 'Y' precio: 5.
ventas vender: producto2 a: cliente.
```