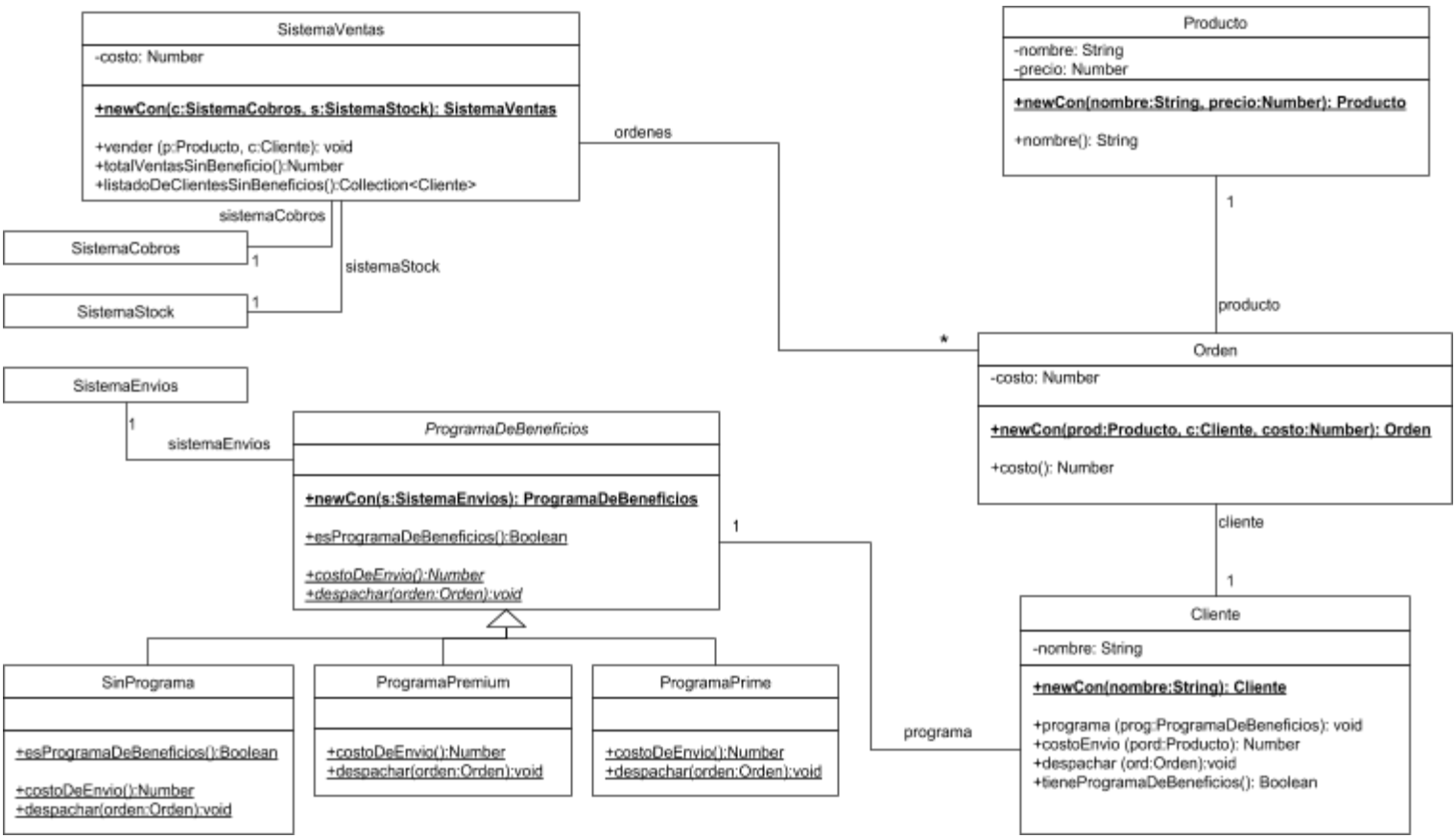
UNLP - Programación Orientada a Objetos 1 2018 - Redictado de 2017
Examen Parcial - Segunda Fecha - 19 de junio de 2018
Solución “en papel” diseñando Cliente con clase representando SinPrograma (sin nil)

Diagrama de Clases



Class Cliente

v.i: ‘ nombre programa ’ “un cliente puede tener muchos otros datos como direccion para el envío, tarjetas de crédito para el cobro, a los efectos de lo que se precisa para el SistemaDeVentas no es necesario y por ello se acepta que no definan en la solucion en papel”

“En esta solucion el sistemaDeEnvios pasa a formar parte del ProgramaDeBeneficios evitando tener que pasarlo como parametro donde hace falta, esto es indistinto de manejar con nil el programa o no. Es una cuestión de diseño, para ver como queda escrito el código de una forma y otra, comparar con la solución publicada con-nil”

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

#initialize
  “En lugar de dejarla en nil, agrego un objeto que encapsula todo el comportamiento relacionado con un Cliente no vinculado a ningún programa. Me evita tener que chequear por nil en todos los mensajes donde debo delegar en el programa de beneficios. Para entender la diferencia ver la solucion publicada con-nil “

  programa := SinPrograma new.
```

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

```
#nombre: aString
  nombre := aString.
  ^self

#programa: unPrograma
   Programa := unPrograma.

#costoEnvio: unProducto
   ^programa costoEnvio: unProducto

#despachar: unaOrden
   ^programa despachar: unaOrden

#tieneProgramaDeBeneficios
"El concepto de estar o no vinculado a un programa es únicamente binario. Por ende un mensaje como este permite hacer cómputos externos a un Cliente sin preocuparnos por si hay o no programa, ni cuantos diferentes programas se manejan. Ver la clase SinPrograma mas abajo y ver la solucion publicada con-nil para comprender las diferencias"

  ^ programa esProgramaDeBeneficios
```

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
```

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

```
#costo
  ^costo
```

Class abstract ProgramaDeBeneficios

```
v.i: 'sistemaEnvios'

class #newCon: unSistemaEnvios
    ^ self new initialize: unSistemaEnvios

#initalize: unSistemEnvios
   sistemaEnvios := unSistemaEnvios.

#sistemaEnvios
    ^sistemaEnvios

#esProgramaDeBeneficios
"Permite diferenciar comportamiento en el Cliente de acuerdo a si esta vinculado o no a un
programa de beneficios. Por defecto toda subclase de ProgramaDeBeneficios representa un
vinculo con un programa.

   ^true

abstract #costoDeEnvio: unProducto

abstract #despachar: unaOrden
```

Class ProgramaDeBeneficios subclass: ProgramaPrime

```
v.i: ''

#descuentoPrime
   ^0.5

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

#despachar: unaOrden
   ^ sistemaEnvios despacharEnvioUrgente: unaOrden

#tieneProgramaDeBeneficios
   ^true
```

Class ProgramaDeBeneficios subclass: ProgramaPremium

```
v.i: 'sistemaEnvios'

class #newCon: unSistemaEnvios
    ^ self new sistemaEnvios: unSistemaEnvios

#sistemaEnvios: unSistemEnvios
```

```
   sistemaEnvios := unSistemaEnvios.

#costoDeEnvio: unProducto
   ^ 0

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

Class ProgramaDeBeneficios subclass: SinPrograma

```
v.i: ‘’

#esProgramaDeBeneficios
“Esta clase permite encapsular el comportamiento del cliente cuando no esta vinculado a ningún
programa de beneficios sin que el Cliente deba preguntar por nil en cada caso”

   ^false

#despachar: unaOrden
“Un Cliente que no está vinculado a ningún programa de beneficios despacha en forma normal”
   ^ sistemaEnvios despacharEnvioNormal: unaOrden

#costoDeEnvio: unProducto
^ sistemaEnvios cotizarEnvio: unProducto
```

Class SistemaVentas

```
v.i: ‘sistemaDeCobros sistemaDeStock ordenes’

class #newConCobros: unSistemaCobros stock: unSistemaDeStock
   ^ self new initConCobros: unSistemaCobros stock: unSistemaDeStock

#initConCobros: unSistemaCobros stock: unSistemaDeStock
  sistemaCobros := unSistemaCobros.
  sistemaDeStock := unSistemaDeStock.
  ordenes := OrderedCollection new.
  ^self

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

       orden := Orden newConProducto: unProducto
                    cliente: unCliente
                      costo: costoTotal.
       sistemaDeCobros cobrar: orden.
```

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

```
        unCliente despachar: orden.
        ordenes add: orden.
        ^orden
    ]
```

#totalVentasSinBeneficios

```
"Retorna la suma de todas las ordenes hechas a clientes sin programa de beneficios  "
^ (ordenes select:[:orden| orden
                              cliente tieneProgramaDeBeneficios 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 tieneProgramaDeBeneficios not] )
              asSortedCollection:[:c1 :c2 | c1 nombre < c2 nombre]
```

Playground

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

ventas := SistemaDeVentas newConCobros: sistemaCobros
                          stock: sistemaDeStock.
cliente := Cliente newConNombre: 'Juan'.
producto1 := Producto newConNombre: 'X' precio: 10.
ventas vender: producto1 a: cliente.
cliente programa: Prime newCon: sistemaEnvios.
producto2 := Producto newConNombre: 'Y' precio: 5.
ventas vender: producto2 a: cliente.
```