Redictado de Orientación a Objetos 1 – 2013
Examen – tercera fecha

Enunciado

La municipalidad de La Plata emite bimestralmente un impuesto denominado Servicios Urbanos Municipales (SUM) para inmuebles (casas, departamentos, locales comerciales, etc.). Este impuesto agrupa varios **conceptos** tal como *Tasas por Servicios Urbanos* y *Contribución por Pavimento* entre otros. La suma de estos conceptos determina un total parcial que debe abonar el propietario del inmueble denominado: **cuota**. Por cada inmueble se conoce la siguiente información: titular de la propiedad, dirección, provincia, numero de partida, y metros cuadrados del mismo. El monto de los conceptos nombrados está dado por un valor de referencia multiplicado por los metros cuadrados del inmueble. Para resolver el valor de referencia usted puede utilizar una clase Referencias que conoce el mensaje #valorReferenciaParaConcepto: unConcepto que puede utilizar en el cálculo.
El valor final del impuesto está definido por la modalidad de pago que realice el contribuyente: Primer Vencimiento, Segundo Vencimiento, y Pago Anual. El valor final del impuesto dentro del primer vencimiento es del valor de la cuota y un posible descuento del 10% si las cuotas anteriores del mismo año fueron pagadas todas dentro del Primer Vencimiento. Un pago denominado Segundo vencimiento tiene como valor final el valor de la cuota más un 3% por recargo. Finalmente, la modalidad de Pago Anual se calcula el valor de las cuotas pendientes de pagar del año y al total se le aplica un descuento del 15%.
Algunos ejemplos de pagos, si fuese una cuota de 100 pesos:

- Pago de la cuota 1/6 a término: $100
- Pago de la cuota 2/6 a término: $90
- Pago de la cuota 3/6 a dentro del segundo vencimiento: $103
- Pago anual (cuotas 4,5, y 6): $255

Cada cuota se emite teniendo como primer vencimiento el día 11 de cada mes y el segundo vencimiento el día 20 de cada mes. No se aceptan pagos realizados fuera de este período y la deuda es tratada por fuera del sistema (no deben ser contemplado).
Cada bimestre se emite la cuota con la siguiente información:

- Información del inmueble
- Número de cuota
- Deuda de cuotas anteriores (de años anteriores y el mismo año)
- Valores de cada modalidad de pago: Primer Vencimiento, Segundo Vencimiento y Pago Anual.

Consigna:

1. Diseñe y represente un modelo UML de clases de su aplicación
2. Implemente completamente en Smalltalk la solución al problema anterior. Implemente lo necesario para calcular las **cuotas del impuesto, generar el resumen y registrar los pagos.**
3. Instancié en el workspace el ejemplo esquematizado en el enunciado.

Ayuda: Recuerde que puede construir una fecha utilizando la siguiente sentencia en base a un dia, mes y año:
Date newDay: day month: monthName year: year

Redictado de Orientación a Objetos 1 – 2013
Examen – segunda fecha

Enunciado

Una **compañía** de telefonía brinda servicios para **líneas móviles** y **fijas**. A las **fijas** se les cobra siempre $0,50 el minuto de llamada, mientras que cada **línea móvil** de la **compañía** tiene un **plan** asociado el cual conoce los costos de cada llamada. Una **línea móvil** podría cambiar el plan asociado en distintos momentos.
El plan **prepago** posee un crédito tope para poder utilizar y un costo por minuto de $1 para llamadas a cualquier línea; una vez alcanzado el tope máximo no es posible realizar otras llamadas.
El servicio **normal** tiene un costo de $0,75 a **líneas móviles** y $0,50 a **líneas fijas** por minuto.
Una línea con plan **flota** abona $0,75 a **líneas móviles** y $0,25 a **líneas fijas** por minuto. Además, tiene asociado un conjunto de líneas a las cuales la llamada no tienen costo.
Todas las **llamadas** entre líneas son registradas y todos los meses la **compañía** calcula el consumo total de cada línea para imprimir un informe. El informe lista, para cada línea, el número, el tipo de plan, todas las llamadas, los minutos totales consumidos y los minutos totales ahorrados en el caso de ser un plan flota.
Por controles estadísticos cada **línea** debe poder calcular, cuando se requiera, la llamada de mayor costo entre todas las llamadas realizadas desde esa **línea**.

A continuación se resume la tabla tarifaria para líneas móviles:

	Móvil	**Fijo**
Prepago	$1	$1
Normal	$0,75	$0,5
Flota	$0,75	$0,25

El resumen generado para una línea móvil con plan flota, qué realizó una llamada de 2 minutos a una línea fija, una llamada de 4 minutos a una línea móvil dentro de su conjunto de líneas asociadas y una llamada de 4 minutos a una línea móvil no incluída en su conjunto de líneas asociadas; tiene la siguiente forma:

```
Línea: 221-1566455
Plan: Flota
Nro destino        Tiempo       Costo
221-1234567        2 min        $0,5
221-1566467        4m           $0
221-1566477        4m           $6
Monto total a pagar: $7,5
Minutos sin costo: 4
```

Consigna:

1. Diseñe y represente un modelo UML de clases de su aplicación.
2. Implemente completamente en Smalltalk la solución al problema anterior. Tenga en cuenta los siguientes mensajes.

```
Compañía>>generarResumen
"Genera el resumen indicado en la sección anterior donde se lista las llamadas
realizadas. Asuma que sólo existen las llamadas del último mes."

Línea>>llamadaMayorCosto
"Retorna la llamada de mayor costo entre todas las llamadas de la línea."

Línea>>llamarA:otraLinea por: cantidadDeMinutos
"Mensaje enviado para establecer una llamada."
```

3. Instancie en el workspace el ejemplo esquematizado en el enunciado.

En una ruta existen estaciones de peajes. La estación está formada por varias cabinas. Los vehículos (motos, autos y camiones) llegán al peaje y deben elegir una cabina, sumándose a su cola.
Las cabinas pueden ser de pago exacto, con vuelto y de camiones. Las cabinas de pago exacto sólo admiten a quienes disponen del pago justo. Las cabinas de camiones sólo admiten camiones. Estas restricciones no impiden que un camión ingrese por la cabina de con vuelto o pago exacto (si lo tiene), o que un vehículo con pago exacto pase por la cabina de con vuelto. Los valores que deben abonar, dependen del vehiculo y de la cabina.

	Pago exacto	Con vuelto	Camion
Moto	1	1.5	
Auto	2	2.5	
Camion	16	8.5	8

El vehículo (suponga que no hay conductor) tiene el dinero para pagar. El dinero puede ser exacto o no. Las cabinas además llevan el registro de los vehículos que pasaron como así también del importe abonado.
-Realice un diagrama de clases (indicando herencia, relaciones de conocimiento, cardinalidad y responsabilidades) a partir del problema planteado para resolver el problema de asignar a un auto a una cabina de peajes con menos vehículos en su cola de espera.
-Realice el/los diagramas de secuencia para asignar un auto a una cabina y cobrarle peaje.
-Defina los mensajes necesarios para crear e inicializar una estación de peajes, con cabinas de peajes.
-Implemente los siguientes mensajes

EstacionDePeaje>> asignarCabinaAUnVehiculo: unVehiculo
" encola al vehicula en alguna cabina"
Cabina>>admiteVehiculo: unVehiculo
"retorna true si la cabina adminte a ese vehiculo"
Cabina>>precioDePeajeParaUnVehiculo: unVehiculo
"retorna la cantidad a pagar por unVehiculo en el receptor"
Cabina>>cobrarAVehiculo: unVehiculo
"cobra el monto que unVehiculo debe pagar"
EstacionDePeaje>>cabinaQueRecaudoLaMayorCantidadDeDinero
"retorna aquella cabina con mayor recaudacion acumulada"
EstacionDePeaje>>cantidadTotalDeAutosQueParonPorTipo
"Retorna cuantos autos han pasado de cada tipo por el receptor"
Cabina>>cantidadTotalDeAutosQuePasaronPorTipo
"Retorna cuantos autos han pasado de cada tipo por el receptor"

Orientación a Objetos I 09/08/12

Ej. 1. La Verificación Tecnica Vehicular (VTV) es un control del funcionamiento de las partes de los vehiculos, que los habilita a circular por un año. En la VTV se revisan diferentes componentes de los vehículos de acuerdo a su tipo (autos, motos y camionetas), esto significa que a cada vehiculo se le testean diferentes componentes.
En la planta de verificacion existen varias vias donde los vehiculos se encolan para ser verificados.
Para agilizar el tramite, existen verificaciones rápidas, que se atienden por las llamadas vias rapidas. En estas vias se verifican menos componentes del vehiculo que en la via normal, pero la verificacion tiene una vigencia de 6 meses (contra 1 año de la normal). Cualquier vehiculo puede ir a cualquier via, el dueño decide por cual ir.
En la via normal los componentes que se verifican para cada vehiculo son los siguientes:
Auto: frenos, emision de gases, direccion. **Moto**: frenos, direccion. **Camioneta**: frenos, emision de gases, direccion, suspension.
En la via rapida los componentes que se verifican para cada vehiculo son los siguientes:
Auto: frenos, direccion. **Moto**: frenos. **Camioneta**: frenos, direccion, suspension.
Se debe implementar un sistema en el cual se simula la verificacion del mismo. Dado un vehiculo que entra a una de las vias, se debe generar un comprobante que indique el tipo de vehiculo, patente, que componentes han sido testeados y la vigencia de la verificacion (dependiendo de la via en que se realizó). Por ejemplo, para una moto que va por la via rapida, la impresión del comprobante (#printOn:) es:
Vehiculo: Moto
Patente: qwe123
Verificacion Frenos
Vigencia: 6 meses

a. Realice el diagrama de clases UML.
b. Realice un diagrama de secuencia para la verificacion de un auto en la via normal.
c. Implemente en Smalltalk.

Ej.2. Indique para cada una de las siguientes afirmaciones si son verdaderas o falsas. Considere el polimorfismo para todos los mensajes publicos de un objeto.

a. Dos objetos polimorficos siempre son instancias de la misma clase.
b. Dos clases polimorficas deben siempre pertenecer a la misma jerarquia
c. Si dos objetos son polimorficos es posible intercambiarlos sin que se produzcan errores del tipo *Mensaje no entendido.*

Orientacion a Objetos I
Parcial - 7 de Diciembre de 2012

Imagine una red de alumbrado compuesta de farolas y sensores de luz para los barrios de una ciudad. Cada barrio tiene su propia red de farolas.

Cada farola está conectada a una o varias vecinas formando un grafo conexo. El conocimiento entre farolas vecinas es mutuo. Cada farola tiene conectado un sensor de luz que permite a la farola saber si es de día o de noche. El sensor de luz se representa con la clase LightSensor, la clase define un mensaje de instancia #isDark que retorna un booleano según este oscuro (true) o no este oscuro (false)

Las farolas entienden los siguientes mensajes
#redEncendida
"retorna true si todas las farolas de un barrio están encendidas"

#estadoNocturno
"retorna true si alguno de los sensores (incluido el propio) reportan que es de noche"

#encender
"enciende la farola receptora y propaga el mensaje a las vecinas"

#apagar
"apaga la farola receptora y propaga el mensaje a las vecinas"

Como se dijo antes, cada barrio tiene una red inteligente de farolas. Todas las redes están controladas por un ControlDeLuminaria que conoce una farola de cada barrio. El ControlDeLuminaria entiende los siguientes mensajes:

#estadoNocturno
"retorna true si alguna de las redes reporta que es de noche"

#controlarLuminarias
"se encienden o apagan las farolas según sea de noche o de dia"

Resuelva:
1. Diagrama Clases
2. Diagrama de Sequencia para el caso de una farola (que conoce a otras dos farolas) cuando recibe el mensaje #estadoNocturno
3. Implemente en Smalltalk
a) métodos mencionados y los invocados que son parte de su modelo
b) constructor / inicializacion de las clases representado farola y ControlDeLuminaria
4. Codigo Workspace que crea un ControlDeLuminaria con una red de dos farolas

Orientación a Objetos I - Curso 2011 - Parcial
Sábado 26 de Noviembre

Se desea modelar redes de sensores inalámbricos. Las mismas están compuestas por dos tipos diferentes de elementos: **Sensores** y **Hubs.**

Cada componente de la red conoce a sus vecinos (la relación es simétrica) a los que les puede mandar mensajes. Los sensores guardan un estado (encendido / apagado), mientras que los Hubs no, ya que sólo retransmiten mensajes a sus vecinos. Además, los Sensores y los Hubs poseen una fecha de fabricación y también conocen una posición que se expresa con una instancia de la clase Point de Smalltalk.

Se desea modelar e implementar la funcionalidad de Encendido / Apagado de toda la red y ciertas consultas sobre la misma. Tanto los sensores como los hubs deben responder al siguiente protocolo:

`#switchOn`
"Si el sensor está apagado, se enciende y propaga el mensaje a los vecinos.
Un Hub siempre propaga el mensaje de encendido"

`#switchOff`
"Si el sensor está encendido, se apaga y propaga el mensaje a los vecinos.
Un Hub siempre propaga el mensaje de apagado"

`#fechaDefabricaciónDelElementoMasViejo`
"Retorna la fecha mas antigua de entre todos los sensores y hubs"

`#posicionesDeElementos`
"Retorna una colección de puntos, que son las posiciones de todos los elementos (sensores / hubs) de la red."

Considere que: (i) la red es una componente conexa, es decir, no hay sensores / hubs que no se conectan con la red. (ii) La posición de cada sensor / Hub es única.
(iii): Los Hubs nunca están linkeados con otro Hub
Considere el siguiente template para declarar sus clases y métodos.

```
<Superclass> subclass: <NombreDeMiClase>
instanceVariables: '<variable1> <variable2>'

MyClass>>esteEsUnMetodoDeInstancia
   "implementación del método"

MyClass  class>> esteEsUnMetodoDeClase
   "implementación del método"
```

Tareas

1. Realice el diagrama de clases UML
2. Realice el diagrama de secuencia en donde se le envía el mensaje #switchOn al sensor1 de la red representada por sensor1<->hub1<->sensor2. Considere que ambos sensores están apagados.
3. Implemente en Smalltalk las clases y los métodos necesarios para cumplir con el protocolo definido, como así también para instanciar redes.
4. Escriba el código Smalltalk necesario para instanciar la red del punto 2 en un workspace.

Parcial Objetos 1

Los programas son administrados por un objeto de la clase: AdministradorDeProgramacion.
Existen otras clases que son parte del diseño.
Usted es el encargado de diseñar e implementar las siguientes operaciones de la clase AdministradorDeProgramacion y las clases necesarias para hacerlo funcionar:

`#retrasarPrograma: unPrograma horas: unNumero`
"Retrasa el programa dada la cantidad de horas indicada por unNumero. todos los programas que siguen ven reflejado el retraso"

`#estirarPrograma: unPrograma horas: unNumero`
"El programa dado se estira la cantidad de horas indicada por unNumero (su hora de finalizacion se retrasa) y retrasa el resto de los programas del dia"

`#programasQueSiguenA: unPrograma`
"Retorna una colección con todos los programas del día que vienen después de unPrograma."

`#programaQueSeEmiteALas: unaHora`
"Retorna el programa que se emite en la hora indicada por el parámetro."

`#programasDesde:unaHora hasta:otraHora`
"Retorna una colección con los programas que se emiten en el rango indicado por los parámetros."

Tareas:

1. Construya un diagrama UML de clases para representar el diseño completo.
2. Implemente completamente en Smalltalk .
3. Muestre en un workspace cómo instancia el ejemplo provisto anteriormente y luego cómo utiliza al AdministradorDeProgramacion para estirar el partido 1 hora.

Resuelto

Orientación a Objetos 1 – Primer Parcial

Ejercicio 1:

Sea una compañía que brinda el servicio de comunicación telefónica para sus abonados, los cuales, pueden poseer varias líneas contratadas. Cada línea cuenta con una cantidad de minutos a utilizar por mes. Esta cantidad se conoce como crédito y se restablece el primero de cada mes.

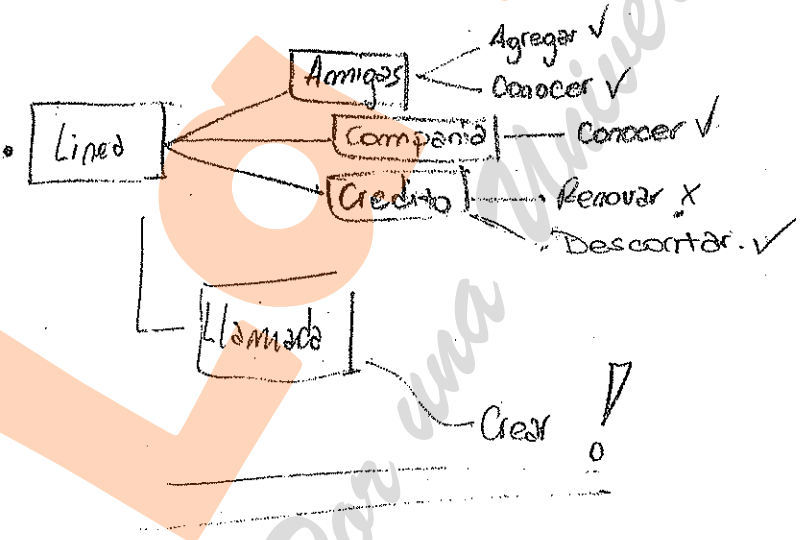
La forma en que se descuenta el crédito es la siguiente. Cuando se realiza una llamada desde una línea a una línea de otra compañía, se reduce el crédito en una cantidad correspondiente a la duración de la llamada. Si en cambio la llamada es a una línea de la misma compañía, se consume sólo el 50 % de la duración de la llamada. Finalmente, cada línea puede tener varias líneas amigas. Las llamadas a las líneas amigas son gratuitas.

Ante una llamada, la compañía determina su mecanismo de descuento de crédito e instancia el objeto apropiado. Para lograr esto, existen las clases LlamadaInterCompañia, LlamadaIntraCompañiaNormal o LlamadaIntraCompañiaNumeroAmigo. Estos objetos saben descontar el crédito correspondiente. A continuación se describen los métodos que se corresponden con la estrategia indicada.

```
>>Compañía>realizarComunicacionDesde: unaLinea hacia: otraLinea conDuracion: minutos
" La compañía crea una instancia de LlamadaInterCompañía, LlamadaIntraCompañíaNormal,
LlamadaIntraCompañíaNumeroAmigo, todas subclases de Llamada"

>>Llamada>descontarCredito
"Reduce el crédito de la línea que realizó la llamada"
```

1. Realice un diagrama de clases de una solución para descontar el crédito de las líneas en función de las llamadas, considerando los métodos indicados.
2. Realice un diagrama de secuencia que comience con el mensaje realizarComunicacionDesde: Hacia: conDuracion: y muestre como se descuenta el crédito para una llamada de 10 minutos a una línea amiga.
3. Instancie en un workspace los objetos necesarios para disparar la situación descripta en el punto anterior.
4. Implemente la solución diseñada en los puntos 1 y 2 en Smalltalk.



1/2

Orientación a Objetos I - Curso 2011 - Parcial
Sábado 26 de Noviembre

Se desea modelar redes de sensores inalámbricos. Las mismas están compuestas por dos tipos diferentes de elementos: **Sensores** y **Hubs.**

Cada componente de la red conoce a sus vecinos (la relación es simétrica) a los que les puede mandar mensajes. Los sensores guardan un estado (encendido / apagado), mientras que los Hubs no, ya que sólo retransmiten mensajes a sus vecinos. Además, los Sensores y los Hubs poseen una fecha de fabricación y también conocen una posición que se expresa con una instancia de la clase Point de Smalltalk.

Se desea modelar e implementar la funcionalidad de Encendido / Apagado de toda la red y ciertas consultas sobre la misma. Tanto los sensores como los hubs deben responder al siguiente protocolo:

`#switchOn`
"Si el sensor está apagado, se enciende y propaga el mensaje a los vecinos.
Un Hub siempre propaga el mensaje de encendido"

`#switchOff`
"Si el sensor está encendido, se apaga y propaga el mensaje a los vecinos.
Un Hub siempre propaga el mensaje de apagado"

`#fechaDefabricaciónDelElementoMasViejo`
"Retorna la fecha mas antigua de entre todos los sensores y hubs"

`#posicionesDeElementos`
"Retorna una colección de puntos, que son las posiciones de todos los elementos (sensores / hubs) de la red."

Considere que: (i) la red es una componente conexa, es decir, no hay sensores / hubs que no se conectan con la red. (ii) La posición de cada sensor / Hub es única.
(iii): Los Hubs nunca están linkeados con otro Hub
Considere el siguiente template para declarar sus clases y métodos.

```
<Superclass> subclass: <NombreDeMiClase>
instanceVariables: '<variable1> <variable2>'

MyClass>>esteEsUnMetodoDeInstancia
   "implementación del método"

MyClass  class>> esteEsUnMetodoDeClase
   "implementación del método"
```

Tareas

1. Realice el diagrama de clases UML
2. Realice el diagrama de secuencia en donde se le envía el mensaje #switchOn al sensor1 de la red representada por sensor1<->hub1<->sensor2. Considere que ambos sensores están apagados.
3. Implemente en Smalltalk las clases y los métodos necesarios para cumplir con el protocolo definido, como así también para instanciar redes.
4. Escriba el código Smalltalk necesario para instanciar la red del punto 2 en un workspace.

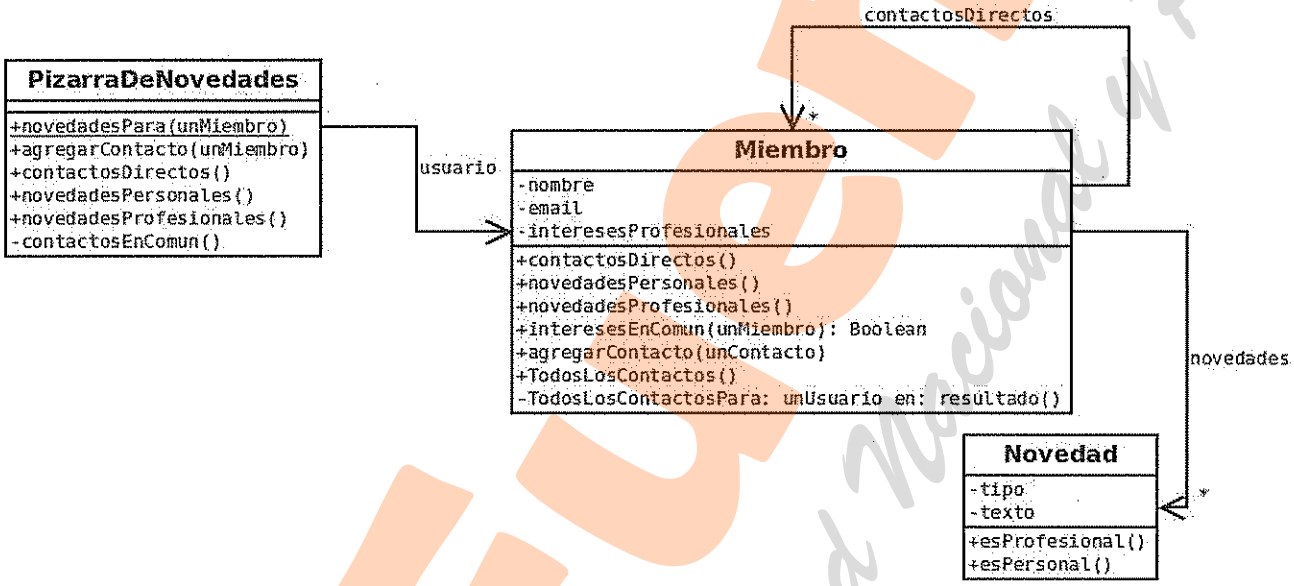
contacto indirecto pero tienen intereses en común (a ambos les interesa #educacion)

1. Documente utilizando Diagrama de Clases UML.
2. Presente el Diagrama de Secuencia UML del envío del mensaje del constructor de PizarraNovedades
3. Considerando los Casos 1, 2 y 3 cuál sería el resultado del siguiente código Smalltalk?

```
(PizarraNovedades novedadesPara: usuarioXavier)
novedadesProfesionales
```

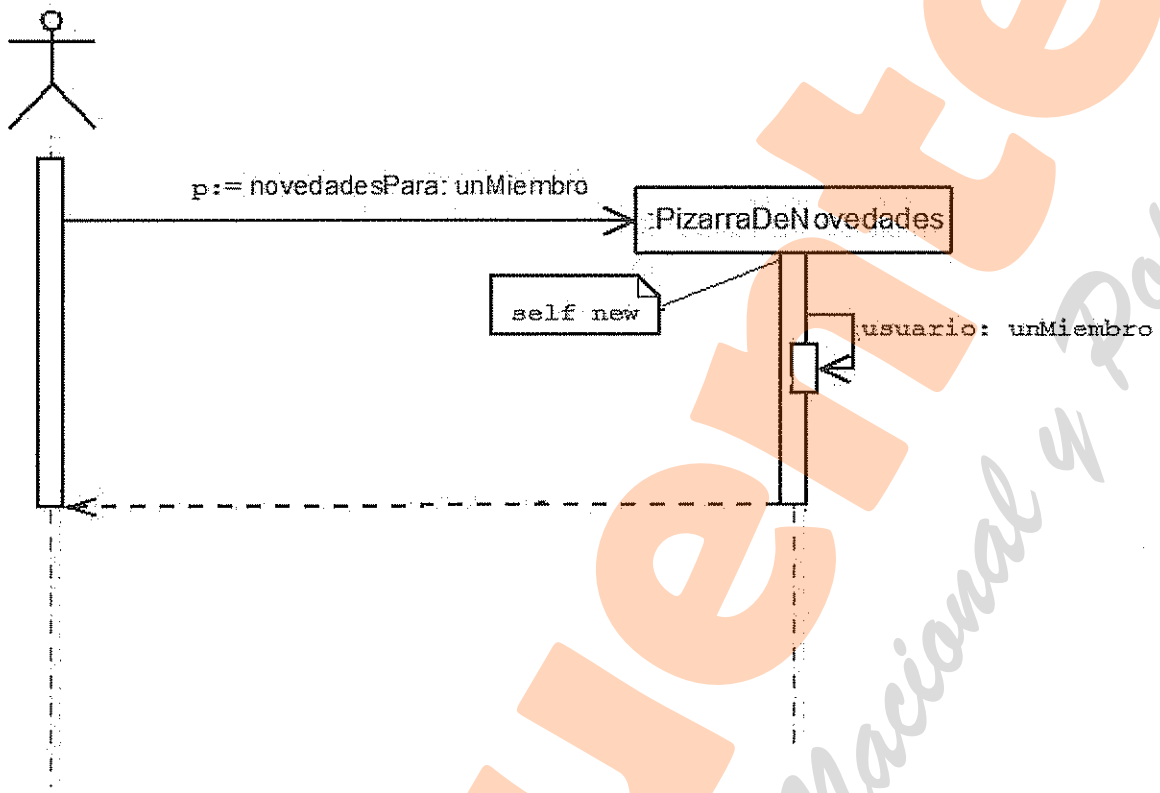
4. Implemente en Smalltalk los métodos de PizarraNovedades y los métodos que estos invocan

Ejercicio 1



Obs: si bien se podría plantear una solución utilizando una jerarquía de Novedad, a los efectos del enunciado, es suficiente la solución planteada.

Ejercicio 2



Ejercicio 3

Devuelve una colección vacía

Ejercicio 4

Pizarra

```
Pizarra class#novedadesPara: unUsuario
```

"toma un usuario como parámetro y retorna una nueva instancia de PizarraNovedades"

```
^ (self new)
     usuario: unUsuario ;
     yourself
```

```
#agregarContacto: unMiembro
```

"Crea el contacto directo bidireccional entre unMiembro y el usuario de esta pizarra"

```
     self usuario agregarContacto: unMiembro.
```

```
#contactosDirectos
```

"retorna la lista ordenada alfabéticamente de contactos directos del usuario de esta pizarra"

```
  ^ self usuario contactosDirectos
```

```
"Retorna si el usuario tiene intereses en común con otroUsuario"
     ^ self intereses anySatisfy:
          [:interes| otroUsuario intereses includes: interes]

#todosLosContactos
"Retorna una colección con todos los contactos (directos e
indirectos) del usuario"
|contactos|
 contactos := OrderedCollection new.
 self contactosDirectos do:[ :c|
      c todosLosContactosPara: self en: contactos.].
 ^  contactos


#todosLosContactosPara: unUsuario en: resultado
"Recorre la red de contactos guardando en resultado, todos los
contactos directos e indirectos de unUsuario"

 (unUsuario ~= self) &  ((resultado includes: self) not)
     ifTrue: [
         resultado add:self.
         self contactosDirectos do:[ :cd |
             cd todosLosContactosPara: unUsuario en:resultado.]]
```

Novedad

```
#esPersonal
 ^ tipo == #personal

#esProfesional
 ^ tipo == #profesional
```

```
#novedadesPersonales
```
"retorna la lista de novedades personales de cada contacto directo"
```
 |novedades|
  novedades:= OrderedCollection new.
  ^ self usuario contactosDirectos do:[:c| novedades addAll:
                                        (c novedadesPersonales)].
     ^novedades

#novedadesProfesionales
```
"retorna la lista de novedades profesionales de contactos directos o indirectos con igual interés"
```
 | novedades |
     novedades := OrderedCollection new.
     self contactosEnComun do: [ :c | novedades addAll:
                                          c novedadesProfesionales].
     ^ novedades

#contactosEnComun
```
"Retorna una colección con los contactos directos e indirectos con intereses comunes a los del usuario "
```
     ^ (self usuario todosLosContactos)
         select:[:c| c interesesEnComun: usuario]
```

Miembro

```
#initialize: unNombre mail: unMail
     nombre := unNombre.
     mail := unMail.
     intereses:= OrderedCollection new.
     novedades := OrderedCollection new.
     contactosDirectos := SortedCollection sortBlock: [:cont1
                                :cont2| cont1 nombre < cont2 nombre].
     ^self

#agregarContacto: unMiembro.
     self contactosDirectos add: unMiembro.
     unMiembro contactosDirectos add: self.

#contactosDirectos
     ^ contactosDirectos

#novedadesPersonales
   ^ self novedades select:[:unaNovedad| unaNovedad esPersonal]

#novedadesProfesionales
   ^ self novedades select:[:unaNovedad| unaNovedad esProfesional]


#interesesEnComun: otroUsuario
```

Orientación a Objetos I
Curso 2013
Miércoles 18 de Diciembre

Sea un sitio para aprender idiomas, en particular, para practicar escritura. Los usuarios registrados pueden: escribir post en el idioma que desean practicar, y/o corregir post escritos en su idioma nativo. Cuando se registran en el sitio indican los idiomas que desean practicar, su idioma nativo y su nivel como corrector: novato o experto (esta categorización se utiliza para sumar puntos cuando corrigen como se describirá más adelante). De esta forma, los usuarios escriben posts en el idioma que desean practicar y hacen correcciones de post escritos en su idioma nativo.

Ejemplo 1: usuarios registrados
John es novato habla inglés y desea practicar español.
Juan es novato habla español y desea practicar inglés y sueco.
Ingrid es experta habla sueco y desea practicar español.
Carlos es experto habla español y desea practicar inglés.

Los posts son puramente texto, en donde se identifican cada una de las oraciones. Además, un post pertenece a un usuario, está escrito en un idioma y posee una fecha.

Ejemplo 2: Post de John
John escribió en español el 13/dic/2013.
1. Ola amigos!
2. Espero ayudarlos con el Inglés mientras mejoro el Español!

Cuando un usuario corrige un post puede agregar comentarios a cada una de las oraciones (pueden quedar oraciones sin comentarios).

Ejemplo 3: Correcciones al post de John realizada por Juan
Juan corrige el post de John.
1. Ola amigos! -> comentario: ¡Hola amigos! es la versión correcta.
2. Espero ayudarlos con el Inglés mientras mejoro el Español! -> no hay comentario

Ejemplo 4: Correcciones al post de John realizada por Carlos
Carlos corrige el post previo.
1. Ola amigos! -> comentario: Hola de saludo se escriben con 'H'.
2. Espero ayudarlos con el Inglés mientras mejoro el Español! -> comentario: Los gentilicios en español se escriben en minúscula.

Finalmente, cada corrección suma puntos para el usuario que hizo la corrección. El puntaje se calcula de la siguiente forma. Cuando corrige un usuario novato, cada oración corregida equivale a 10 puntos. Si corrige más de 5 oraciones en el mismo post, se duplica el puntaje. Cuando corrige un usuario experto, cada oración equivale a 12 puntos. Y además si el puntaje es mayor o igual que 24, suma tantos puntos como oraciones corregidas en el post.

Ejemplo 5:
La corrección de Juan equivale a 10 puntos. Ya que Juan es novato y corrigió sólo 1 oración (y esto es menos de 5 oraciones).

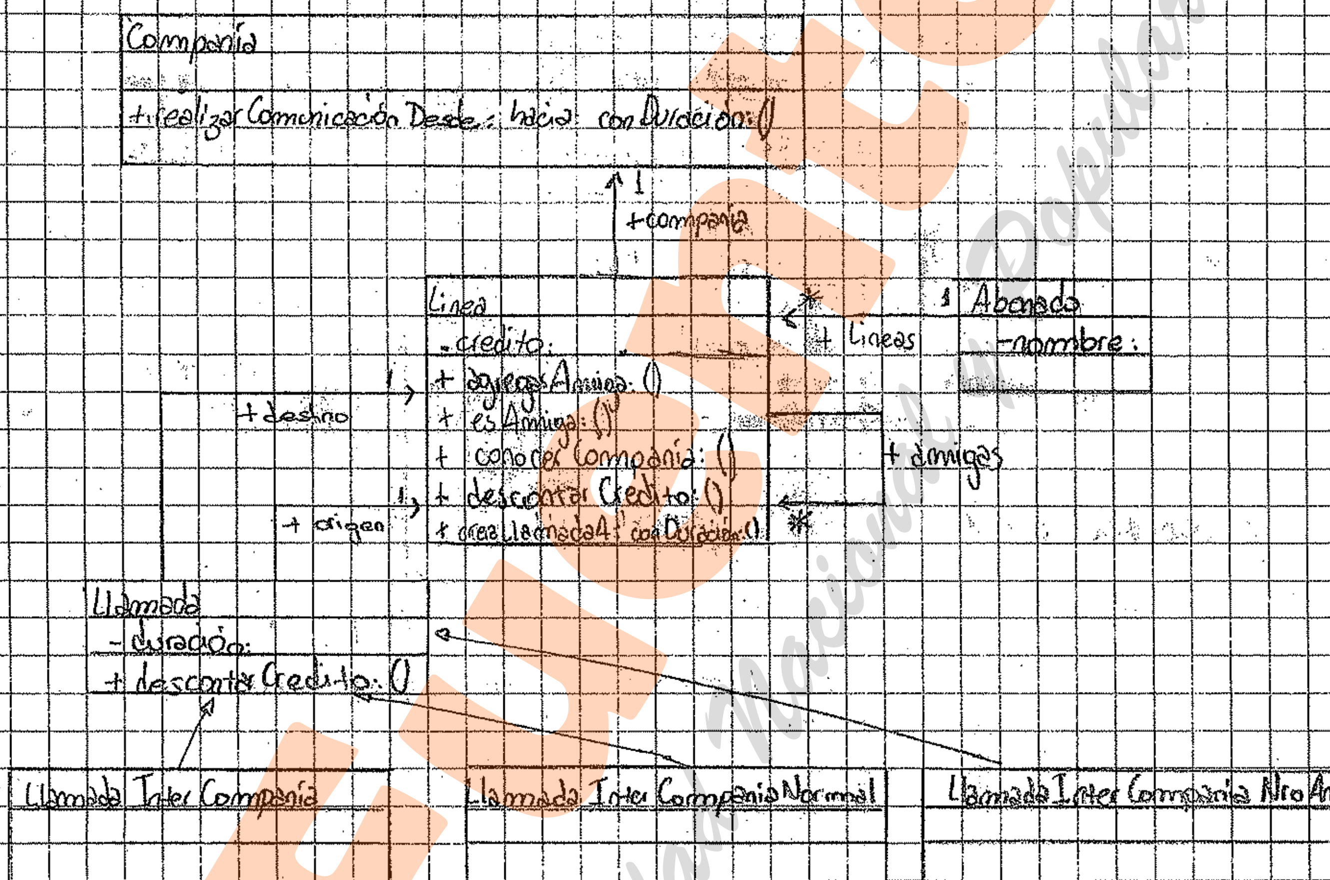
Ejemplo 6
La corrección de Carlos equivale a 26 puntos. Ya que Carlos es experto, así que cada oración equivale a 12 puntos y como suma 24, suma 2 puntos más.

El sistema debe permitir realizar la siguiente funcionalidad:
- Instanciar un usuario novato o experto. Indicando su nombre, su idioma nativo y los idiomas que desea aprender.
- Crear un post. Indicando fecha, usuario y texto. Para el texto considere que recibe una colección donde cada elemento es una oración.
- Crear una corrección. Indicando fecha, post que corrige, usuario y las correcciones. Tenga presente que cada corrección debe relacionarse con una oración.
- Retornar todos los posts que un usuario podría corregir. Este método necesita recibir un usuario y retorna los posts escritos en su idioma nativo, pero que no corrigió. Por ejemplo, si el usuario indicado es Carlos, se retornaría el post del ejemplo 2. Si en cambio el usuario es Juan, no existe post escrito en español que no haya corregido.
- retornar el post más popular. Es decir, el que posee más correcciones.
- retornar el post más reciente. Es decir, el más nuevo.
- retornar la oración más popular. Es decir, la que más correcciones posee. Cabe señalar que la oración más popular no necesariamente se corresponde con el post más popular. Para el ejemplo 2, la oración más popular es la primera (Ola amigos!)
- retornar los puntos de un usuario.

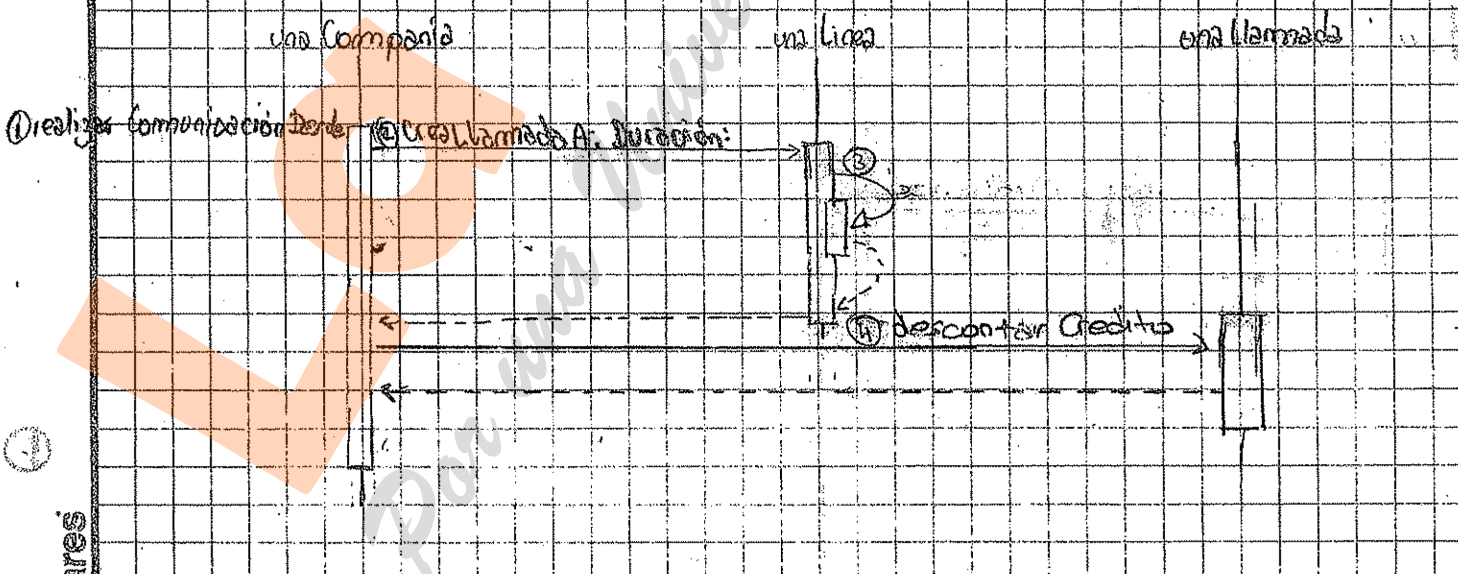
Tareas:
1.- Realice un diagrama de clases
2.- Implemente la funcionalidad indicada
3.- Escriba un workspace para instanciar el Ejemplo 3.

Parcial 1er fecha 2010:

1. Diagrama de clases:



2. Diagrama de Secuencia:



4. Smalltalk:

Crear LlamadaA: otraLinea duración: minutos

(self esAmigo: otraLinea)
ifTrue:
[^ llamadaInterCompañíaNuevaAmigo → descontar crédito
newOrigen: self
duración: minutos
newDestino: otraLinea].
(self mismaCompañía: otraLinea) "Nada"
ifTrue:
[^ LlamadaInterCompañíaNormal → descontar crédito.
newOrigen: self
duración: minutos
newDestino: otraLinea]. (self duración / 2) ceiling
?
^ LlamadaInterCompañía → descontar crédito
newOrigen: self
duración: minutos
newDestino: otraLinea.

self origen: descontarCredito:
self duración

3. Workspace:

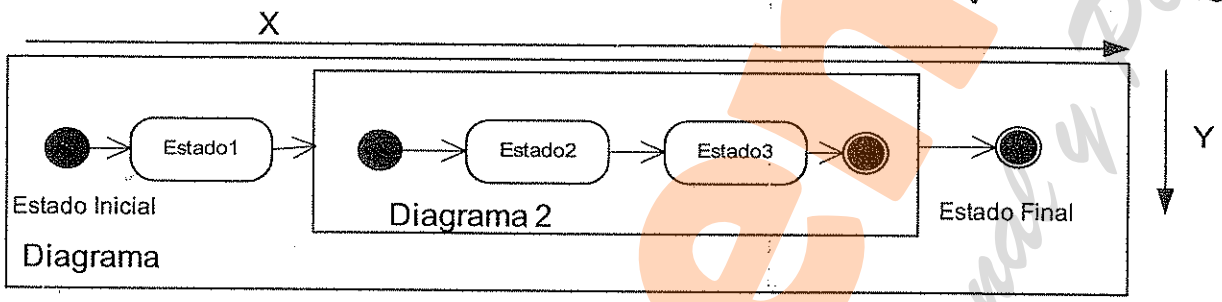
|Llamada|

llamada := unaLinea CrearLlamadaA: otraLinea duración: minutos
llamada descontarCredito:
^llamada

Redictado de Orientación a Objetos 1 – 2014
Examen – primer fecha

Enunciado

Se desea modelar una herramienta flexible y escalable que permita asistir la tarea de diseñar gráficamente diagramas de actividad UML. La herramienta permitirá especificar sobre un lienzo elementos que pueden ser (transiciones o elementos) tal como estado simple, estado inicial (estado que inicia una actividad), y estado final (estado que finaliza una actividad). **Cada diagrama conoce un único estado inicial.** (SIEMPRE EXISTE)



Los elementos en un lienzo tienen las características típicas de las figuras geométricas, como por ejemplo alto, ancho y posición en el espacio. Las transiciones conocen una figura origen y otra destino. Cada elemento tiene un punto (X@Y) donde se ubican en el lienzo. Los estados (simples) conocen su altura y ancho mientras que los estados (iniciales) y (finales) conocen su radio por ser graficados con un círculo. Por último, para facilitar la manipulación de varios elementos, se permiten definir (agrupación) como subdiagramas. Por ejemplo, un diagrama puede definir un conjunto de estados con sus transiciones.

Modele una solución y provea la funcionalidad para consultar:

>Diagrama# superficie
``Retorna el punto de la figura más alejada en el lienzo representando el tamaño del mismo. Para su cálculo se contempla la posición del elemento más alejado en el eje x y del elemento más alejado en el eje y. Recuerde que el mensaje **#@** permite definir un punto y los mensajes **#x** e **#y** permiten acceder a cada valor de coordenada correspondientemente"

> Diagrama# cantidadDeEstados
"retorna el total de estados conocidos directa e indirectamente por el diagrama"

>Elemento# esFinal
"retorna verdadero si el estado es final y falso si no lo es"

> Diagrama# finaliza
"returna verdadero si el diagrama posee al menos un estado final y falso si no posee estados finales"

>Elemento# esInicial
"retorna verdadero si el estado es inicial y falso si no lo es"

>Elemento# clonar
"retorna una copia del elemento actual. Si es estado simple, crea una instancia nueva y copia sus transiciones. Si es una transición, clona los elementos de destino." (Y ORIGEN)

>Diagrama# clonar
"retorna una copia idéntica al diagrama con nuevas referencias a objetos"

>Elemento# agregarTransicionAEstado: unEstado → ELEMENTO O DIAGRAMA
"Agrega una transición desde el estado receptor del mensaje al estado recibido por parámetro. Los estados finales al recibir este mensaje no agregan la transición."

>Diagrama# imprimir
"imprime en el Transcript la secuencia de estados. Para cada estado se imprime su nombre, para transiciones una flecha '->' y para subdiagramas el contenido entre corchetes () respetando las pautas anteriores"
Para el diagrama de ejemplo, se debería mostrar:
EstadoInicial -> Estado1 -> (EstadoInicial ->Estado2 -> Estado3 ->EstadFinal) -> EstadoFinal"

Consigna:

1. Diseñe y represente un modelo UML de clases de su aplicación
2. Implemente completamente en Smalltalk, incluyendo constructores e inicializadores
3. Instancié en el workspace el ejemplo esquematizado en el enunciado

Notas:

- Para imprimir en el transcript, puede enviar el mensaje #show:
  Transcript show: 'Un string'

- Para definir las clases utilice el siguiente template:
  <superclass> subclass: <MyClass>
  <MyClass> instanceVariables: '<variable1> <variable2>'

- Defina las clases utilice el siguiente template:
  "Método de clase"
  MyClass class>>classMethod

  "Método de instancia"
  MyClass>>instanceMethod

HACER:

1 SETTER

1 GETTER

Orientacion a Objetos I

Parcial - 15 de Febrero 2013

Ejercicio 1

Una empresa de instalación y reparación de equipos de aire acondicionado centrales genera Presupuestos que están conformados por una colección de Items. Los Items que conforman un Presupuesto pueden ser Materiales (mangueras, cinta, caños, gas) y Mano de Obra. De los materiales se debe registrar: descripción y monto. De la Mano de Obra se debe registrar: rubro (electricidad, transporte, electrónica, instalación), horas estimadas, valor de la hora, cantidad de personas.

Cada Item del presupuesto tiene un costo agregado según se aplique, o el IVA o el cargo de ART (Aseguradora de Riesgo de Trabajo). El IVA es un impuesto del 10% que se aplica a los Items de Materiales. El cargo de ART se aplica a los Items de Mano de Obra, se calcula como $100 * cantidad de personas.
Por ejemplo, el presupuesto para reparar un aire acondicionado en un edificio incluye: un caño de bronce que cuesta $200 y 2hs de un electricista que cuesta $100 por hora. El importe total de presupuesto será:

Costo de Caño * (1 + 10%) + (Costo electricista + ART de una persona)

Se espera que en el diseño que Ud. presente, los objetos representando Items respondan los siguientes mensajes:
#costoNominal "costo sin considerar ni IVA, ni ART"
#costoAgregado "el costo de IVA o ART según corresponda"
#costoTotal "costo nominal + costo agregado"

Se espera que objetos representando Presupuestos respondan los siguientes mensajes:
#totalPresupuesto "retorna el total del presupuesto"
#cantidadItems "responde la cantidad de items del presupuesto"
#itemMasCaro "retorna el Item con mayor costo total"
#costoIVA "retorna la sumatoria del IVA del presupuesto"
#agregarHorasHombre:unMonto costoHora: unValor cantidadPersonas: unaCantidad rubro: desc
"Invoca el constructor de item adecuado y agrega el item al presupuesto"
#agregarMaterial: descripcion monto: unValor
"Invoca el constructor de item adecuado y agrega el item al presupuesto"

Documente su solucion presentando:
1. Diagrama de Clases

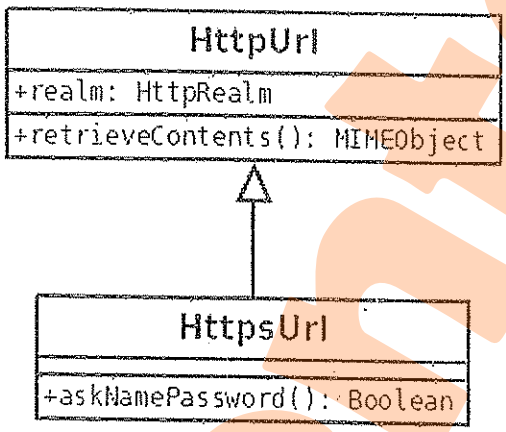
2. Diagrama de Secuencia cuando una instancia de Presupuesto recibe el mensaje #totalPresupuesto, considerando el ejemplo presentado en el enunciado.

3. Implemente en Smalltalk las clases de su diseño y los métodos que se mencionan en el enunciado

4. Presente un workspace para el caso de un Presupuesto que incluya: 2hs de un electricista a $100 por hora y un caño de bronce con valor $200. Finalmente, se muestra el #totalPresupuesto.

Ejercicio 2

A partir del diagrama de clases conteste las siguientes preguntas:



a) Instancias de HttpsUrl acceden al atributo **realm** como variable de instancia propia. (Verdadero/Falso)

b) Instancias de HttpUrl entienden el mensaje **askNamePassword()**. (Verdadero/Falso)

c) Instancias de HttpUrl y HttpsUrl, ¿son polimórficas? ¿Por qué?

d) ¿Cuál es el tipo de retorno de la operación **retrieveContents()**?

e) El atributo **realm** es público. ¿Explique cómo lo implementaría en Smalltalk? (conteste brevemente)

Redictado de Orientación a Objetos 1 – 2014
Examen – segunda fecha

Enunciado

Un supermercado requiere un nuevo sistema de facturación que permita aplicar promociones. El supermercado posee varias cajas registradoras encargadas de procesar las compras de los clientes. Como es habitual, cada cliente visita los diferentes pasillos del supermercado seleccionando productos de diferentes tipos. Cada producto se distingue con un número identificador también conocido como código de barra, un nombre, contenido (por ejemplo 20gm o 1Litro), marca, cantidad en stock, descripción, precio y su categoría. Las categorías de productos son lácteos, carnes, harinas y otros.

Cuando el cliente finaliza su recorrido en el supermercado se dirige a una de las cajas registradoras para realizar el pago de los productos. En la caja se solicita información tal como el medio de pago y los productos para generar un ticket de la compra. El monto total de la compra se computa sumando el precio de todos los productos adquiridos y luego se aplican las promociones vigentes. A una compra se asocia un medio de pago previamente a la aplicación de las promociones. Se disponen tres tipos de medios de pago: efectivo, tarjeta de débito y tarjeta de crédito. Las tarjetas de crédito y debito poseen un identificador en base a cuatro números, por ej.: 1234 – 1234 – 1234 – 1234. Esta información se utiliza al imprimir el ticket.

Para el procesamiento de una compra se espera un medio de pago y una lista de productos que permite genera un ticket que contiene líneas para cada producto, el descuento de las promociones y el monto total calculado en base a la diferencia de los dos montos anteriores.

NO, PARA (2.)

Las promociones son acumulables y al ser evaluadas reciben una lista de productos como parámetro y retornan un descuento si corresponde. Las promociones pueden ser:

1. 3x2. Por cada 3 productos similares (con el mismo número de código de barra) solo son facturados dos de ellos. Por ejemplo, si encontramos 5 gaseosas iguales sólo se cobran 4.
2. Promociones por medios de pagos: el supermercado brinda promociones relacionadas con el medio de pago. Si la compra se abonará con tarjeta de débito el descuento es del 10%, si es con tarjeta de crédito del 15% y si la compra se abona en efectivo el descuento es del 5% del total.
3. Marca Nac&Pop. El gobierno decidió promover los productos de una nueva marca nacional llamada "Nac&Poc". Si la compra incluye al menos un producto de esta marca, se le realizará un descuento por el 5% del total de los productos.
4. Promo Lácteos: el supermercado ofrece una promoción a partir de la cual se descuenta 10% en todos los productos lácteos.

Las promociones pueden ser activables dependiendo la estrategia comercial. Por ejemplo, un fin de semana se activan las promociones 1 y 2 pero otro fin de semana pueden ser activadas las promociones 2 y 4. El sistema debe permitir esta flexibilidad.

Se requiere además que se imprima el ticket de la siguiente forma (lo indicado en negrita es la parte dinámica):

MiSupermercado
Medio de pago: ***Débito – número de tarjeta 1234 – 1234 - 1234 - 1234***
Lista de productos:

Pan fargo	***600g***	***$30***
Pan fargo	***600g***	***$30***
Pan fargo	***600g***	***$30***
Pan Nac&Pop	***200g***	***$10***

Monto total sin descuentos: $100

Descuentos:

DESCUENTO)	***Pan Fargo 3x2***	***-$30***
	Nac&Pop	***-$5***
	Pago débito	***-$10***

Total: ***$55***

Si el medio de pago es en efectivo, simplemente se indica "Efectivo".

Consigna:

1. Diseñe y represente un modelo UML de clases y un diagrama de secuencia para el procesamiento de los productos.
2. Implemente completamente en Smalltalk la solución al problema anterior.
3. Instancié en el workspace el ejemplo esquematizado en el enunciado.
    - De ejemplos de cómo configurar la caja registradora para activar la promoción 1 y 3, y luego otro ejemplo para las reglas 2 y 4.

Notas:

- Para imprimir en el transcript, puede enviar el mensaje #show:

Transcript show: 'Un string'

- Para definir las clases utilice el siguiente template:

<superclass> subclass: <MyClass>
<MyClass> instanceVariables: '<variable1> <variable2>'

- Defina las clases utilice el siguiente template:

"Método de clase"
MyClass class>>classMethod

"Método de instancia"
MyClass>>instanceMethod