Orientación a Objetos 1 - 2016
Práctica 5

Ejercicio 1

Sea una red social en donde los usuarios tienen un muro en el cual pueden agregar mensajes para luego ser leídos por todos. La red también incluye relaciones de amistad entre los usuarios, lo que permite que los usuarios sean avisados cuando un amigo hizo una publicación en su muro.
Implemente la red social con la particularidad de que el muro de cada usuario sólo permite 10 mensajes como máximo (cuando llega el mensaje número 11 se descarta el mensaje más viejo).

```
Wall>>post:aMessage
"Agrega un mensaje al muro"

Wall>>list
"Retorna una colección con todos los mensajes del muro"

Wall>>remove: aMessage
"Elimina el mensaje del muro"

Wall>>numberOfMessages
"Retorna la cantidad de mensajes posteados en el muro"
```

Los usuarios tienen un nombre, el muro y su lista de amigos. Cuando un usuario publica (con el mensaje #post:) algo en su muro, sus amigos deben ser notificados. Considere que la clase que implementa al usuario debe contener el siguiente protocolo:

```
User>>name
“Retorna el nombre del usuario”

User>>name: aString
“Cambia el nombre del usuario por el valor recibido”

User>>addFriend: anotherUser
“Agrega anotherUser a la lista de amigos”

User>>post:aMessage
"Publica un mensaje en su muro"

User>>newMessage:aMessage from:anotherUser
```

"Los usuarios reciben este mensaje cuando el amigo indicado por el parámetro anotherUser publicó el mensaje aMessage en su propio muro"

Tareas

1. Realice un diagrama de Clases.
2. Realice un diagrama de secuencia para mostrar cómo un usuario que tiene dos amigos agrega algo a su muro (y se agrega a los muros de los amigos).
3. Implemente en Pharo.
    3.1 Incluya código de prueba en un Playground.
    3.2 Con la asistencia de un ayudante, implemente un test case en base al código del punto anterior.

Ejercicio 1.1

En la red social del ejercicio anterior se desea implementar la funcionalidad que responde a la premisa: “Los amigos de mis amigos también son mis amigos”.
Implemente el método #twoLevelsPost: el cual es similar a #post, pero le llega “a los amigos de mis amigos".

Ejemplo:
Si Juan es amigo de Pedro, Pedro es amigo de Carlos y sin importar si Juan y Carlos son amigos, cuando Juan hace un twoLevelst:, Carlos debe enterarse.

Tareas

1. Realice un diagrama de secuencia para el ejemplo.
2. Implemente en Pharo.

Ejercicio 1.2

Extienda el ejercicio anterior con el mensaje #floodPost:, en cual tiene una funcionalidad similar a las anteriores pero llega a TODAS las personas alcanzables (amigos, amigos de mis amigos, amigos de mis amigos de mis amigos… y así sucesivamente).

Warning: considere cuidadosamente la posibilidad de entrar en loop infinito.

Ejercicio 2

Diseñe e implemente un cliente de correo electrónico de acuerdo con la siguiente especificación:

Un email tiene como atributos principales la dirección de correo del remitente, la del destinatario, un asunto, un cuerpo y una fecha. El sistema debe almacenar todos los emails que recibe en la bandeja de entrada. Además, la aplicación permite eliminar un mensaje de la bandeja de entrada, y en ese caso, pasa a otra bandeja de la que dispone el cliente de correo, que es la bandeja de eliminados. De la bandeja de eliminados, también se pueden borrar los mails, pero al eliminarlos de allí se borran definitivamente del cliente de correo.

La aplicación debe permitir:

- marcar un email como "leído"
- recuperar (en una colección) todos los emails no leídos de una bandeja.
- determinar el espacio ocupado por una bandeja de entrada (para esto considere que el tamaño de un email se calcula como el tamaño de su cuerpo, siendo el cuerpo un String).
- retornar (en una colección) el campo asunto de los emails de cualquier bandeja.
- retornar los mails de una bandeja ordenados cronológicamente
- retornar los mails de una bandeja ordenados por tamaño.
- eliminar mails de las bandejas respetando el comportamiento explicado anteriormente.

Tareas:

1. Realice el diagrama de clases.
2. Implemente en Pharo.

Ejercicio 3

Sea una empresa telefónica que brinda servicios de comunicación a sus abonados. Las comunicaciones pueden ser locales, interurbanas e internacionales. De cada comunicación se conoce el momento de su comienzo, la distancia entre los destinos (que llama y que recibe) y la duración. Cada una de las llamadas se factura de una forma distinta. Las locales tienen un costo fijo por minuto. Las interurbanas tienen un valor que depende de la ciudad destino y en función de la distancia (hay 3 rangos discriminados) es el costo de la misma. Y por último, para las llamadas internacionales el costo depende de la hora en la que comience la misma. Si comienza entre las 08:00 y las 20:00 tienen un costo, mientras que de noche tienen un costo menor.

Por otro lado, los abonados se clasifican en dos categorías. Están los *particulares* a los que se les factura el precio neto y las entidades gubernamentales que reciben un 10% de descuento.

Tareas:

1. Realice el diagrama de clases.
2. Implemente el mensaje para calcular el monto que cada abonado debe pagar.
3. Implemente el mensaje para calcular la llamada de mayor duración entre todas las llamadas de entre todos los abonados, para las llamadas realizadas dentro de los últimos 30 días.

4. Implemente el mensaje para calcular el abonado con mayor tiempo total de comunicación para las llamadas realizadas dentro de los últimos 30 días.

Ejercicio 4

Una Media Library contiene medios (videos e imágenes) que luego son mostrados de alguna manera.
La función principal de la media library es recolectar y organizar esos contenidos, de forma tal que luego puedan ser recuperados.

Cada medio tiene un nombre o título, una fecha, un tamaño, una serie de tags que permiten organizarlos y una duración en segundos.

Un video además tiene asociado un string que denota el codec de video.

Por ej. un video determinado puede componerse de los siguientes datos
Gol de Maradona a los ingleses, fecha: 22 de Junio de 1986, tamaño 400KB, Tags: futbol, maradona, mundiales, Duración: 73, CodecVideo: h264

Si quieren ver el video (por favor no comparar con Messi, no hay punto de comparación) https://www.youtube.com/watch?v=7404jcxTu0o

La duración de una imagen es 0.

MediaLibrary debe:

1. Permitir el agregado de medios. Considere que no pueden existir medios repetidos.
2. Buscar todos los medios con un determinado *tag*.
3. Buscar todos los medios que contengan al menos un tag de una lista que se le pasa como parámetro.
4. Buscar todos aquellos medios previos a una determinada fecha que se pasa como parámetro.
5. Recuperar todas las imágenes con un determinado tag.
6. Recupere todos los videos de una categoría.
7. Obtener todos los videos que ocupan menor a un parámetro (expresado en KB).

Tareas:

1. Defina el protocolo para la media library.
2. Diseñe, realice el diagrama de clases UML.
3. Implemente en Pharo.
4. En el Playground muestre como instancia 3 medios: 2 películas y 1 una imagen. Luego muestre cómo invoca cada uno de los mensaje solicitados anteriormente.

Ejercicio 4.1

Considere el agregado de un nuevo tipo de medio a la MediaLibrary. El nuevo tipo de música. La música es un medio que tiene asociado un Género y un intérprete.

Considere que ahora la Media Library debe poder:
1- Retornar todos los medios cuyo nombre contiene un string pasado como parámetro como parte de su nombre.

Considere todos los cambios necesarios para proveer la funcionalidad requerida.

Tareas:

1. Defina el protocolo para la media library.
2. Diseñe, realice el diagrama de clases UML.
3. Implemente en Pharo.
4. En el Playground muestre como instancia 3 medios: 2 películas y 1 una imagen. Luego muestre cómo invoca cada uno de los mensaje solicitados anteriormente.

Ejercicio 5

Imagine una red de alumbrado donde cada farola está conectada a una o varias vecinas formando un grafo conexo. Cada una de las farolas tiene un interruptor. Es suficiente con encender o apagar una farola para que se enciendan o apaguen todas las demás. Sin embargo, si se intenta apagar una farola apagada (o si se intenta encender una farola encendida) no habrá ningún efecto, ya que no se propagará esta acción hacia las vecinas.

Tareas:

1. Realice el diagrama de clases.

2. Realice el diagrama de secuencia para el escenario en donde se enciende una farola con dos vecinas que están apagadas y se conocen mutuamente.

3. Implemente en Pharo los siguientes métodos para las farolas:

```
#initialize
"Inicializa a la farola como apagada"

#pairWithNeighbor: otraFarola
"Crea la relación de vecinos entre las farolas. La relación de vecinos entre las
farolas es recíproca, es decir el receptor del mensaje será vecino de otraFarola, al
igual que otraFarola también se convertirá en vecina del receptor del mensaje."

#turnOn
```

```
"Si la farola no está encendida, la enciende y propaga la acción."

#turnOff
"Si la farola no está apagada, la apaga y propaga la acción."

#isOn
"Retorna true si la farola está encendida."
```

4. Implemente el método de instancia #createLightPost en la clase TestLightGrid. Este método debe retornar una instancia de la farola, la cual debe estar inicializada apropiadamente.

5. Utilice los tests provistos por la cátedra para probar las implementaciones del punto 3.

6. Implemente el mensaje #emergencyTurnOn. Este mensaje puede ser enviado a cualquier farola de la red de alumbrado y debe propagarse a toda la red sin importar si las farolas están encendidas o apagadas. Todas las farolas deben recibir el mensaje.

7. Cree un nuevo tipo de farola, la cual al agregarle una vecina se asegura de que el estado de la nueva vecina coincida con el propio.

8. Implemente el método de instancia #createLightPost en la clase TestLightAcuteGrid. Este método retorna una instancia de la nueva farola definida en el punto 6, la cual debe estar inicializada apropiadamente.

9. Indique por qué el test #testMixed es diferente. Discuta este punto con un ayudante.

Ejercicio 6

Sean los semáforos de tránsito de una ciudad que quiere proveer de “onda verde” a los conductores. Para ello, cuando un semáforo recibe la orden de cambiar a verde, espera 20 segundos y le propaga el pedido al siguiente.

Tareas

1. Realice el diagrama de clases.

2. Implemente en Pharo la clase TrafficLight con los siguientes métodos:

#initialize
"Inicializa el semáforo en luz roja"

#pairWithNeighbor: otroSemaforo
"Crea la relación de vecinos entre los semáforos. Tenga presente que a diferencia de las farolas, con los semáforos hay un orden entre ellos."

#green
"El semáforo cambia a verde, espera 20 segundos y le propaga el pedido al siguiente."

Para implementar el retardo puede utilizar una instancia de la clase Delay, por ejemplo:

```
"Código para esperar 3 segundos"
|d|
d:= Delay forSeconds: 3.
d wait.
```

3. Cree un test para verificar que el método #green propaga correctamente la onda verde (no se preocupe por testear el retardo).

Ejercicio 7

Un SpoolerFIFO es un administrador de impresión que maneja una lista de documentos que deben ser impresos respetando el orden en que fueron enviados a imprimir. El protocolo de SpoolerFIFO incluye los siguientes mensajes:

```
#spool: aDocument
"El spooler agrega aDocument en su cola de impresión."

#nextDocument
"Retorna el siguiente documento a imprimir (FIFO) o nil si no hay ninguno."
```

De los documentos se conoce: el nombre del mismo, su contenido y los datos del usuario que lo creó.

Tareas

1. Realice el diagrama de clases.

2. Implemente en Pharo el SpoolerFIFO.

3. ¿Cómo logró que los documentos sean impresos en el orden en que fueron recibidos?

4. Implemente en Pharo un SpoolerLIFO (last in first out) que permite imprimir los documentos en el orden inverso al que fueron enviados a imprimir.

5. Extender el Spooler con un PrioritySpooler que entiende el mensaje: #spool: aDocument withPriority: aPriority, el cual ordena los documentos por prioridad. En este caso hay que tener en cuenta que la información de prioridad es ajena al documento y solamente se utiliza para el spooler de impresión.

6. Implemente en el spooler los siguientes reportes:

```
#sortedDocumentsFromUser: aUsername
"Retorna, ordenados por nombre, los documentos del usuario."

#documentsFromUser: aUsername sortedBy: aBlock
"Retorna los documentos del usuario, ordenados por el criterio recibido como
parámetro."

#documentsGreaterThan: aSize
"Retorna una lista con los documentos de tamaño mayor al recibido, manteniendo
el orden de impresión."
```

Ejercicio 8

En una librería existen dos tipos de libros: libros de texto y de literatura. De ellos se conoce su título, autor, precio y cantidad de páginas. En la librería se organizó una promoción por el comienzo de clases en la cual la facturación varía de acuerdo al tipo y número de libros comprados. Usted debe implementar un facturador para ventas con promociones el cual genera las facturas que poseen los libros que integran la compra, el bruto y el neto facturados.
Las promociones se aplican de la siguiente manera:
Los libros de texto tienen un 20 % de descuento.
Los libros de literatura tienen un 15 % de descuento.
Por la compra de 5 o más libros se realiza un 2 % extra de descuento.
Adicionalmente a todos aquellos clientes que alguna vez compraron algo en la librería se les realiza un 5% extra de descuento (se calcula sobre el neto resultante de aplicar los descuentos anteriores).

Su solución debe respetar las siguientes indicaciones:
La clase Facturador debe entender el mensaje #facturar: unosLibros para: unCliente, que genera la factura correspondiente.
Una vez determinado el monto a pagar, el mismo debe informarse en el Transcript usando el mensaje #show:.
La clase Facturador debe entender el mensaje #totalDescontado que retorna el monto total descontado por las promociones.

Tareas

1. Realice el diagrama de clases.

2. Realice un diagrama de secuencia UML donde se muestre cómo se determina el monto final de una compra por un libro de texto y uno de literatura.

3. Implemente en Pharo.

4. Instancie en un Playground su facturador, 2 libros de texto y 2 de literatura (a su elección), un nuevo cliente y envíe al facturador el mensaje #facturar:para:. Indique cuál es el monto que deberá informarse en el Transcript para ese caso en particular.

Ejercicios Avanzados

Ejercicio 9

Una instancia de Homero responde a los siguientes mensajes

```
#beberCerveza
"imprime en el Transcript: 'bebiendo una Duff'"

#verTelevision
"imprime en el transcript: 'Mirando sala de emergencia para
monos'"
```

Cuando recibe cualquier otro mensaje un Homero, responde en el Transcript con el text "D'oh!".

Tarea

1. Estudie el método #doesNotUnderstand en Object, categoría "reflective operations".
2. Implemente el Smalltak el objeto Homero.
3. Compruebe que las instancias de Homero responden correctamente.
4. Extienda la implementación para que además de imprimir: D'oh! en Transcript indique el nombre del mensaje que no entendió. Por ejemplo si en el playground escribimos:

   ```
   Homero new irAlTrabajo
   ```

   El resultado en el Transcript debería ser:

   ```
   irAlTrabajo ? Do'h!
   ```

Ejercicio 10

Inspeccione a *nil* (utilice la opción "inspect it" desde un Playground) y responda:

1. ¿A qué clase pertenece?
2. ¿Qué comportamiento se implementa en el testing de esa clase?
3. ¿Puede instanciar esa clase?

Ejercicio 11

Ahora que conoce un poco más sobre *nil* defina el siguiente comportamiento para nil. El objeto nil ahora debe comportarse como el elemento neutro para la suma y la multiplicación cuando aparece como receptor del mensaje + o * respectivamente. Es decir, si en playground evaluamos:

```
nil + 9
```

el resultado debe ser 9. Y si en playground evaluamos

```
nil * 25
```

el resultado debe ser 25.

Tareas

1. Implemente en la clase de la cual es instancia nil, los mensajes + y *, que reciben como parámetro un número.
2. Pruebe en el Playground.

Ejercicio 12

En Smalltalk todo son objetos: las clases y los métodos también. Enviando el mensaje #methods a una clase obtendrá un array con todos los métodos. Y enviando el mensaje #withAllSubclasses a una clase obtendrá una colección con todas las subclases de esa clase.

Tareas

1. Busque la implementación de ambos métodos y lea los comentarios para entender la funcionalidad provista. Sugerencia: utilice el finder, que se puede encontrar dentro del tools en el menú de World.
2. Utilizando esos mensajes indique cuántos métodos se encuentran definidos en la jerarquía de Collection.