Orientación a Objetos 1 - 2012
Práctica 10

Ejercicio 1

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 tiene ningún efecto, es decir no propaga la acción hacia las vecinas.

Tareas:

1. Realice el diagrama de clases UML .
2. Realice el diagrama de secuencia para el escenario en donde se enciende una farola con dos vecinas. Tenga en cuenta que las vecinas se conocen mutuamente.
3. Implemente en Smalltalk los siguientes métodos para las farolas:

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

#pairWithNeighbor: otraFarola
"Crea la relación de vecinos entre las farolas. Tenga presente que la
relación de vecinos entre las farolas es reciproca, es decir el
receptor del mensaje es vecino de otraFarola, al igual que otraFarola
también es vecina del receptor del mensaje "

#turnOn
"Si la farola no esta encendida, la enciende y propaga la acción"

#turnOff
"Si la farola no esta apagado, la apaga y propaga la acción"

#isOn
"Retorna true si la farola esta 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 test provistos por la cátedra para probar las implementaciones del punto 3.
6. Cree un nuevo tipo de farola, la cual al agregarle una vecina se asegura que el estado de la nueva vecina coincida con el propio.
7. Implemente el método de instancia `#createLightPost` en la clase `TestLightAcuteGrid`. Este método retorna una instancia de la nueva farola definida en 6), la cual debe estar inicializada apropiadamente.
8. Indique por qué el test `#testMixed` es diferente. Discuta este punto con un ayudante.

Orientación a Objetos 1 - 2012
Práctica 10

Ejercicio 2

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 UML.
2. Implemente en Smalltalk los siguientes métodos:

```
#initialize
"Inicializa el semaforo en luz roja"

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

#green
"El semaforo 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 3

A partir de la resolución de los ejercicios 1 y 2, conteste a las siguientes preguntas:

1. Para que las farolas conozcan a sus vecinas o para que el semáforo conozca al siguiente, ¿organizó a los elementos en un grafo o en una lista?
2. Según los haya organizado en un grafo o en una lista, ¿qué objetos son los vértices y aristas del grafo, y cuáles son los nodos de la lista?
3. ¿Necesitó modelar explícitamente las estructuras de datos?

Orientación a Objetos 1 - 2012
Práctica 10

Ejercicio 4

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

```
#spool: aDocument
"El spooler encola aDocument en su cola de impresion"

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

Tareas:

1. Realice el diagrama de clases UML .
2. Implemente en Smalltalk el `SpoolerFIFO` .
3. ¿Cómo logró que los documentos sean impresos en el orden en que fueron recibidos?
4. Implemente en Smalltalk 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.

Ejercicio 5

Se desea extender el ejercicio 5 de la práctica 9 para realizar mantenimientos sobre las cuentas. Para eso considere las siguientes especificaciones.

Las cajas de ahorro acumulan un interés sobre el saldo de la cuenta y no permiten realizar extracciones por montos mayores al saldo disponible (leer más abajo sobre el costo de las extracciones). Las cuentas corrientes mantienen su límite de descubierto permitido.

Para realizar el mantenimiento sobre las cuentas se requiere registrar mas información relativa a las cuentas. En el momento de crear una instancia de `CajaDeAhorro` se estipula: el saldo inicial, la tasa de interés y el monto fijo de mantenimiento. En el momento de crear instancias de `CuentaCorriente` se estipula: saldo inicial, monto fijo de matenimiento y el tope de descubierto.

Tanto `CajaDeAhorro` como `CuentaCorriente` tienen una colección de movimientos. Existen dos tipos diferentes de Movimientos: `Deposito` y `Extraccion`. Los depósitos manejan los siguientes datos: fecha de realización y monto. Las extracciones manejan los siguientes datos: fecha de realización, monto y costo de extracción.

El costo de extracción es el 2% del monto a extraer. El parámetro de `#extraer:unMonto` es el monto que el usuario desea retirar, pero también debe descontarse el costo de la extracción. Si no se puede realizar la extracción por falta de saldo (para ambos tipos de cuenta), se debe enviar un mensaje de error al `Transcript`.

Orientación a Objetos 1 - 2012
Práctica 10

Cuando se realiza el mantenimiento los diferentes tipos de cuentas realizan distintas acciones:

- Las instancias de `CajaDeAhorro` realizan dos movimientos: un depósito cuyo monto corresponde a la tasa de interés aplicada sobre el saldo actual (la tasa de interés es un porcentaje); y una extracción con un monto fijo de mantenimiento (con su correspondiente costo asociado). Tanto el valor de la tasa de interés como el monto de mantenimiento se estipula para cada cuenta en el momento de creación.
- Las instancias de `CuentaCorriente` realizan una extracción con un monto fijo de mantenimiento. El valor del mantenimiento mensual se fija para cada cuenta.

Al asignar el saldo inicial (para `CajaDeAhorro` y `CuentaCorriente`) se genera el primer movimiento de la cuenta que es un depósito por el valor del saldo inicial.

Considere que su solución debe contemplar los siguientes mensajes para las cuentas :

```
#ultimos20Movimientos

"Retorna una colección con los últimos 20 movimientos ordenados por fecha y monto. Fecha en orden decreciente, cuando las fechas son las mismas por monto en forma ascendente"

#realizarMantenimiento

"Realiza el mantenimiento correspondiente de la cuenta"
```

Tareas:

1. Realice el diagrama de clases UML.
2. Extienda su implementación en Smalltalk del ejercicio 5 de la práctica 9 para soportar los nuevos requerimientos. Implemente todos los métodos necesarios para la creacion/inicializacion, y uso de objetos `CuentaCorriente` y `CajaDeAhorro`.
3. En un workspace, cree una caja de ahorro con un mantenimiento mensual de $1.5, una tasa del 0.5% y con saldo inicial 10.000. Luego muestre como se realiza un mantenimiento mensual (envíe el mensaje que corresponda para iniciarlo).