Programación 4

PARCIAL FINAL EDICIÓN 2013

Por favor siga las siguientes indicaciones:

- Escriba con lápiz
- Escriba las hojas de un solo lado
- Escriba su nombre y número de documento en todas las hojas que entregue
- Numere las hojas e indique el total de hojas en la primera de ellas
- Recuerde entregar su numero de parcial junto al parcial

Problema 1 (30 puntos)

a) ¿En términos de qué se especifican las pre y post condiciones de un contrato de software?

b) **Contexto**: Dada una cierta bonanza económica en la actividad agropecuaria nacional, y en particular en la producción de granos, un amigo suyo decide emprender nuevos horizontes y abrir una empresa de alquiler de cosechadoras. Estos aparatos tienen un costo elevadísimo, pudiendo superar fácilmente los USD 100.000 de costo cada una, lo cual hace que muchos productores pequeños y medianos no puedan afrontar su compra, y deban alquilarlas en las zafras respectivas. Y es aquí en donde su amigo ha visto un potencial de negocios: alquilar cosechadoras a pequeños y medianos productores de granos, principalmente soja.

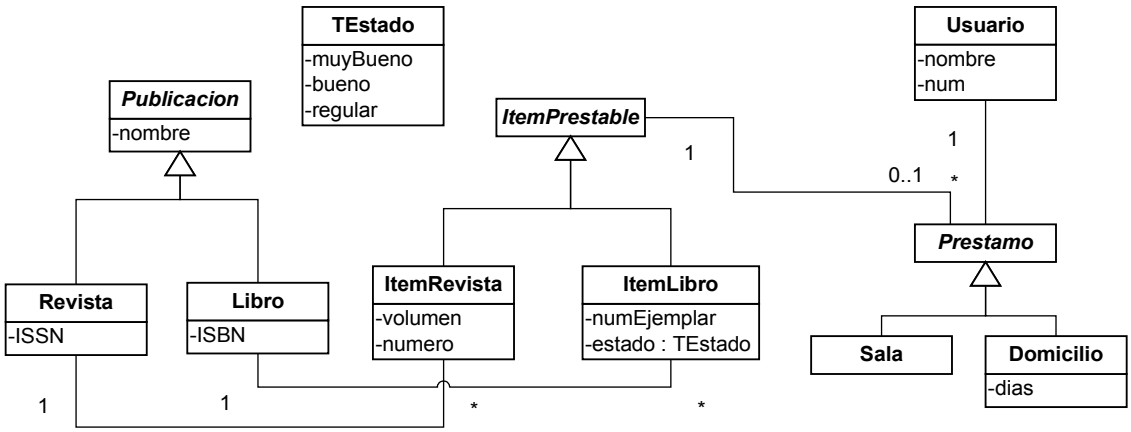
Problema: Por lo tanto, el negocio de su amigo consiste en alquilar una cosechadora (de las varias que ha comprado de diferente marca y modelo) para cosechar cuadros. Un establecimiento rural está compuesto por varios cuadros, cada uno identificado por un código y con un área (medida en hectáreas). Es especialmente importante registrar las "pasadas" que la cosechadora hace en cada cuadro, indicando la fecha de la pasada y la cantidad de toneladas cosechadas. Una misma cosechadora podrá realizar múltiples pasadas por el mismo cuadro, en diferentes fechas, al igual que un cuadro puede ser cosechado por diferentes cosechadoras a lo largo de su vida. Su amigo necesita guardar registro de todas las pasadas que todas sus cosechadoras han hecho sobre los cuadros de los establecimientos, pues a partir de esos datos (fecha y toneladas) podrá luego cobrar a los dueños de dichos establecimientos. Cada cosechadora (que se identifica con un código) es manejada por un operario especialmente entrenado, e interesa también saber qué operario manejó la cosechadora en cada pasada.

Se pide: A partir de la descripción del problema anterior, realice un Modelo de Dominio con restricciones en lenguaje natural.

c) A partir de la descripción del problema anterior, realice un DSS para el Caso de Uso *Ingresar Pasada*, el cual comienza eligiendo un cuadro (identificándolo por su código) de todos los disponibles en el sistema, luego eligiendo la cosechadora (identificándola por su código) de todas las disponibles en el sistema, y finalmente registrando una pasada de esa cosechadora sobre ese cuadro, con la fecha de la pasada y las toneladas.

<u>Problema 2</u> (35 puntos)

La biblioteca de un instituto mantiene información acerca de los libros y revistas que tiene, los usuarios que acceden a dichas publicaciones y los respectivos préstamos. La figura muestra un modelo de dominio del problema, donde las publicaciones representan la descripción de las mismas, mientras que los items prestables representan los volúmenes tangibles que pueden ser prestados a los usuarios. Las instancias de `ItemLibro` correspondientes a un mismo `Libro` se diferencian por el valor de su atributo `numEjemplar` (identificador numérico único autogenerado por el sistema); el atributo `estado` indica el deterioro del ejemplar debido a su uso. Las instancias de `ItemRevista` se identifican por su `volumen` y `numero` (asignados por la editorial de la `Revista` correspondiente); la biblioteca tiene un solo ejemplar de cada (`volumen`, `numero`). Los préstamos pueden ser para `Sala` o `Domicilio`; en el último caso se registra por cuantos días se otorgó. Se desea registrar solamente los préstamos actuales, no el historial de los mismos.



<u>Parte A:</u>

Se consideran los casos de uso *Ingresar nuevo libro* y *Obtener préstamos de un libro*, cada uno de los cuales es modelado con una sola operación del sistema, cuyos contratos se especifican a continuación.

ingrNuevoLibro(nl, isbn : String; es: Set(TEstado))	
Descripción	Da de alta un nuevo libro en el sistema junto con sus ejemplares.

Precondiciones	- No existe en el sistema un `Libro` de nombre `nl`.
Postcondiciones	- Existe en el sistema una nueva instancia de `Libro` cuyos atributos `nombre` e `ISBN` tienen los valores `nl` e `isbn` respectivamente. - Por cada data value `e` en el conjunto `es`, existe en el sistema una nueva instancia de `ItemLibro` cuyo atributo `estado` tiene el valor `e`. El atributo `numEjemplar` tiene el valor del último número disponible para tal fin en el sistema (el cual se actualiza). - Se crea un link entre la nueva instancia de `Libro` y cada una de las nuevas instancias de `ItemLibro`.

darPrestamosLibro(nl : String) : Set (DatosPrestamo)	
Descripción	Devuelve información de todos los préstamos de un libro.
Precondiciones	- Existe en el sistema una instancia de `Libro` identificada por `nl`.
Postcondiciones	Se retorna una colección de datavalues donde cada elemento: - Corresponde a una instancia de `Prestamo`, asociada a un `ItemLibro` correspondiente al `Libro` cuyo nombre es `nl`. - Tiene el nombre del `Usuario` asociado al `Prestamo`, la cantidad de días si es un préstamo a `Domicilio` y el número de ejemplar del `Libro`.

Se pide:

i. Realizar los Diagramas de Comunicación de las dos operaciones descritas, indicando el tipo de visibilidad en todos los mensajes y definiendo el DataType DatosPrestamo.

ii. Realizar el Diagrama de Clases de Diseño resultante.

ParteB:

A los usuarios (así como operadores, administradores u otros individuos que puedan agregarse en el futuro) les interesa enterarse cuando llega un nuevo número de una revista. Para ello se propone que se registren en el sistema, indicando el nombre de la revista que les interesa. Luego, cada vez que se ingresa un nuevo número de la revista, se avisa a los individuos interesados en ella, enviándoles el texto "Llegó número X de revista Y", que se almacenará en una colección de mensajes de cada individuo.

Se pide:

iii. Realizar las modificaciones necesarias al DCD realizado en la parte ii, para diseñar este mecanismo.

iv. Realizar el Diagrama de Comunicación completo de la operación
`Sistema::nuevoNumRevista(nom:String; vol,num:Integer)`
que ingresa al sistema un nuevo número de la revista identificada por `nom` y avisa a los individuos interesados enviando el texto correspondiente.

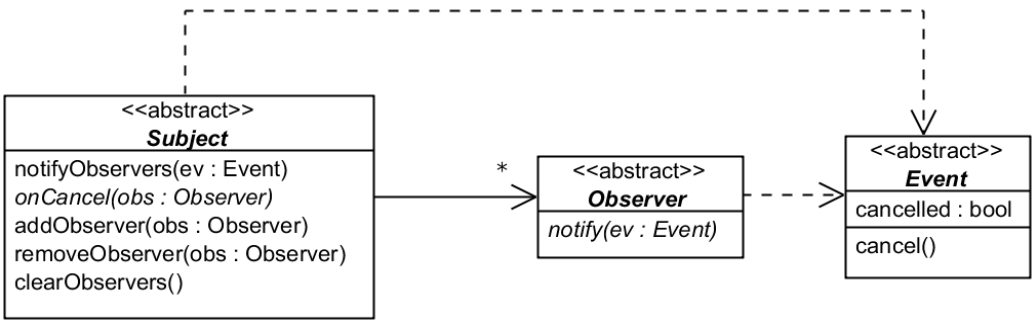
v. En caso de haber usado un Patrón de Diseño, identificarlo indicando los roles de las clases participantes.

Problema 3 (35 puntos)

Observaciones:
- Puede suponer la existencia de implementaciones de `IDictionary`, `ICollection`, `IKey` e `IIterator` según sea necesario.
- Es posible utilizar las clases `set<T>` y `map<K,V>` de la STL.

Parte i

Usted es parte del equipo de desarrollo de un sitio web de compras y se le encomendó la tarea de implementar una variante del patrón Observer para reutilizar luego en la implementación de otros componentes. En esta variante los observadores pueden cancelar la propagación del evento que se notifica. Esto es útil en ciertas ocasiones cuando un observador maneja el evento y no es necesario que los demás observadores sean notificados. El siguiente Diagrama de Clases de Diseño (que deberá respetarse) ilustra las clases que forman el patrón.



Un observador puede cancelar un evento invocando a `ev.cancel()` donde `ev` es el parámetro de la operación `notify(ev: Event)`. Al igual que en el patrón Observer se invoca al método `notify()` cuando ocurre un evento en el Subject. La operación `notify(ev: Event)` no tiene método en la clase `Observer`.

La operación `notifyObservers(ev: Event)`, notifica a los observers en forma secuencial (no importa el orden). Si alguno de ellos cancela el evento, entonces deja de notificar al resto de los observers. Un pseudocódigo de la operación es el siguiente:

```
Subject::notifyObservers(ev: Event):
      Para cada observador obs:
            Se invoca a obs.notify(ev)
            Si ev.cancelled es true:
                  se invoca a this.onCancel(obs)
                  se termina la iteración
```

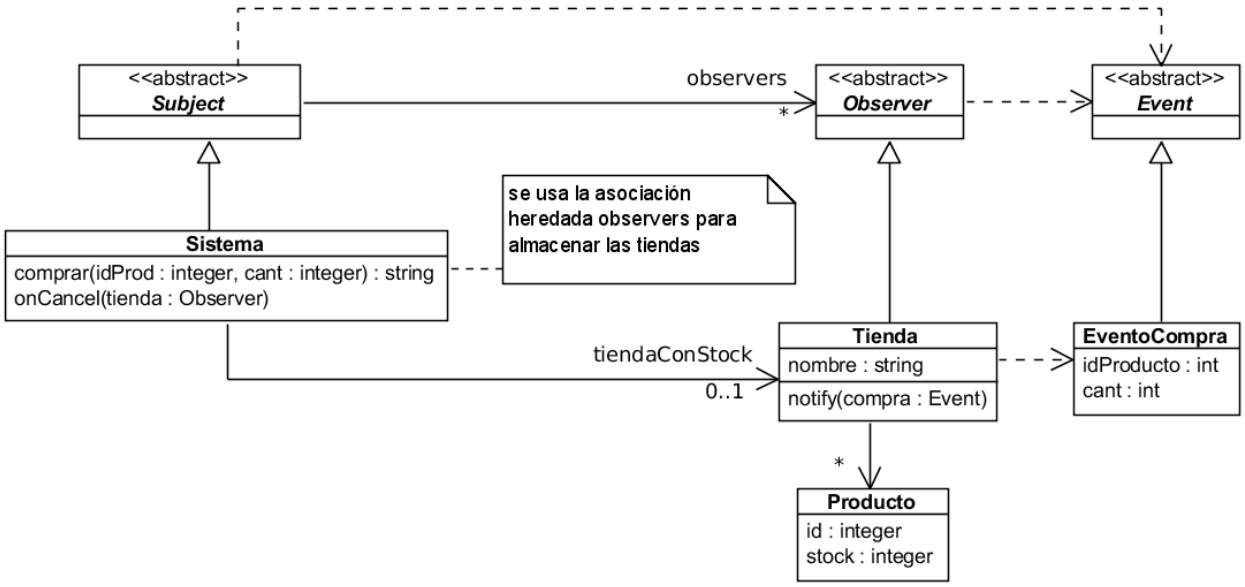
La operación `Subject::onCancel(obs: Observer)` es abstracta y se utiliza como parte de un patrón Template Method para avisar que un observador canceló el evento.

Las operaciones `addObserver(obs: Observer)` y `removeObserver(obs: Observer)` agregan y quitan un observador al subject respectivamente mientras que `clearObservers()` quita a todos los observadores.

Se pide: Implementar las clases `Subject` y `Observer`. No incluir constructores para ninguna de las clases.

Parte ii

Es necesario implementar el mecanismo de compra del sitio web de compras reutilizando el patrón Observer implementado en la parte anterior. La funcionalidad de comprar se encapsula en la operación `comprar(idProd: integer, cant: integer)` de la clase `Sistema`. Al comprar se notifica a todas las tiendas y cada una de ellas inspecciona si hay stock del producto comprado. Si alguna de las tiendas tiene stock del producto que se compra actualiza el stock y cancela el evento para que las demás tiendas no actualicen el stock innecesariamente. La clase `EventoCompra` mantiene la información del id del producto y la cantidad que se compra. El diagrama siguiente muestra el diseño que hay que respetar durante la implementación.



El método `comprar(idProd: integer, cant: integer)` notifica a las tiendas y devuelve el nombre de la tienda en la cuál se hizo la compra, que es la tienda que cancela el evento `EventoCompra`. Si ninguna tienda canceló el evento se lanza una excepción de tipo `invalid_argument`. El pseudocódigo del método es el siguiente:

```
comprar(idProd: integer, cant: integer):string :
      setear tiendaConStock en NULL
      crear un ev = EventoCompra(idProd, cant)
      invocar a notifyObservers(ev)
      si tiendaConStock es NULL:
            lanzar una excepción invalid_argument
      de lo contrario:
            devolver el valor del atributo nombre de
            tiendaConStock
```

La operación `onCancel(tienda: Observer)` setea el atributo `tiendaConStock` con el valor que se pasa por parámetro. Nótese puede ser necesario hacer casts.

El método `notify(compra: Event)` de la clase `Tienda` busca si tiene el producto indicado en el evento; si es así reduce el stock y cancela el evento. Un pseudocódigo para el método es el siguiente:

```
notify(compra):
      si la tienda tiene el producto y hay suficiente stock:
            reducir el stock del producto de id =
            compra.idProducto en compra.cant unidades
            invocar a compra.cancel()
```

Nótese que puede ser necesario hacer casts para convertir el parámetro `compra` a un objeto de tipo `EventoCompra`.

Se pide: Implementar las clases `Sistema` y `Tienda`. No incluir constructores para la clase `Tienda`.