Primera parte. Identificación de casos propuestos

Trabajo asincrónico durante la sesión 1 de la semana 14 (martes 25 de mayo).

Se presentan 5 casos que cubren los primeros patrones de comportamiento que hemos visto durante la semana 13. Los equipos deberán indicar el patrón con que se resuelve TODOS y cada uno de los casos, colocando los detalles de solución en la siguiente ficha:

Problema No. 1

Contexto:

Se debe simular el comportamiento de un servidor que atiende peticiones de manera concurrente. En ciertas ocasiones, como cuando se toma un cierto periodo de descanso, se planea apagarlo por completo, cuya acción a manera interna genera una teminación y desconexión de los servicios activos, se crean uso registros de log y el servidor se apaga de manera controlada. Cuando se retorna del descanso, debe iniciarse de nuevo y esto toma algunos segundos porque debe llevar a cabo el arranque de varios servicios y realizar algunas inicializaciones.

Una vez "levantado" el servidor puede iniciar su proceso de atención de peticiones que provienen de distintos clientes, las cuales son atendida en un tiempo promedio. No obstante, en momentos, poco comunes, el servidor puede entrar en un modo de saturación debido a la cantidad excesiva de peticiones, en este caso, "rechaza" las peticiones entrantes hasta que haya alcanzado un nivel adecuado de manejo y luego vuelve a restablecer la posibilidad de atención.

En cualquier momento, en caso de que ocurra algo inesperado a nivel del entorno físico del mismo, el servidor podría apagarse automáticamente.

Patrón Propuesto: ( ) Comportamiento
State

Diagrama original del patrón:

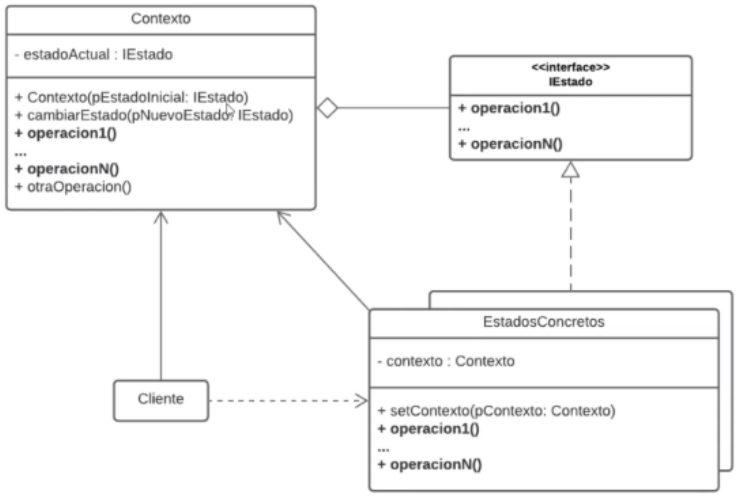
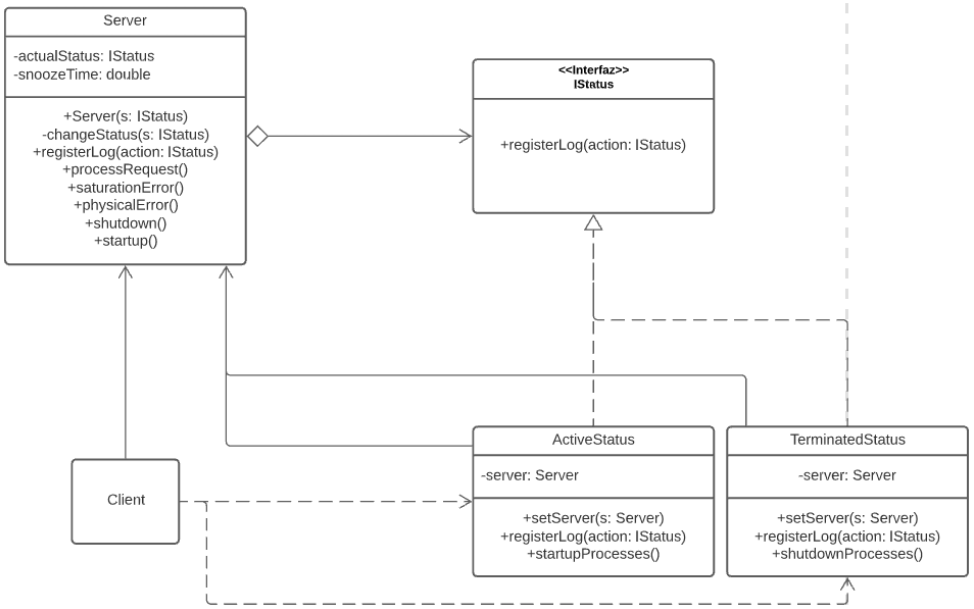


Diagrama del patrón en el contexto del problema:



Rol de cada elemento del patrón en la solución propuesta:

- Server: Es el servidor que recibe las peticiones de manera concurrente de varios clientes. Tiene dos posibles estados, los cuales son ActiveStatus y TerminatedStatus.
- Client: Es el que realiza las peticiones al servidor y estas se procesan dependiendo del estado del server.
- IStatus: Representa los posibles estados del servidor, estos se registran en auditoría.
- ActiveStatus: Representa el estado activo del servidor. Este tiene que ejecutar procesos para iniciar el servidor.
- TerminatedStatus: Representa el estado terminado del servidor. Este tiene que ejecutar procesos para apagar el servidor.

Problema No. 2

Contexto: Una tienda por departamentos ha decido realizar una campaña de descuentos en productos y la mantendrá durante el año aplicando un porcentaje de descuento específico dependiendo de la temporada del año: Navidad, Easter, Año Nuevo. Se desea que el sistema de cobro aplique el descuento establecido de manera automática sin que el dependiente en cajas deba anotar el porcentaje que se ha fijado. La tienda valorará introducir otras épocas o temporadas para agregar nuevos descuentos, como la entrada de verano o invierno.

Patrón Propuesto: ( ) Comportamiento

Strategy

Diagrama original del patrón:

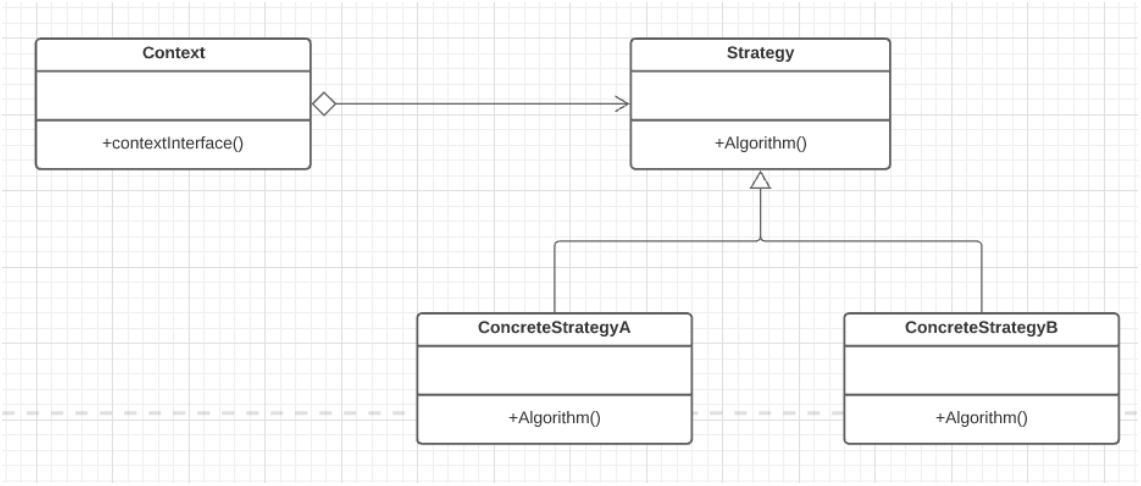
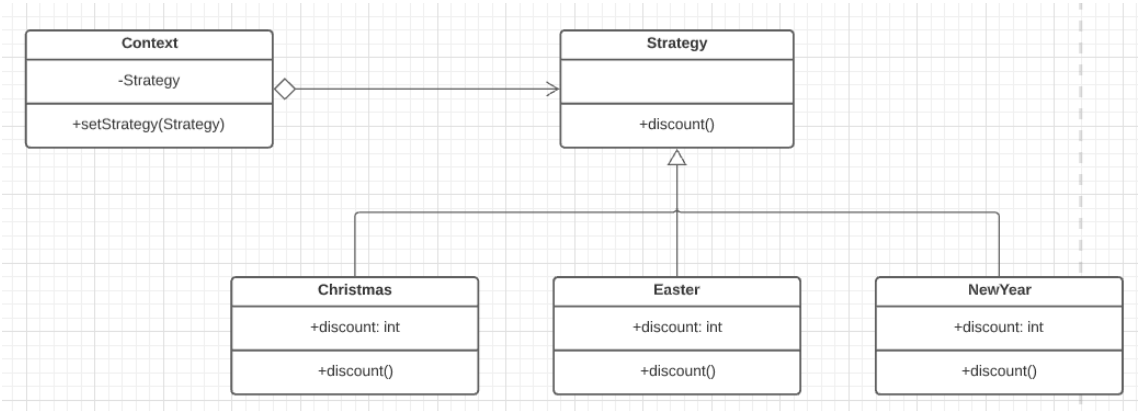


Diagrama del patrón en el contexto del problema:



Rol de cada elemento del patrón en la solución propuesta:

- Context: Encapsula la estrategia a utilizar
- Strategy: Define una interfaz común a los algoritmos que soporta
- ConcreteStrategy: Representa las estrategias concretas
    - Christmas: Contiene la estrategia para la temporada de Navidad
    - Easter: Contiene la estrategia para la temporada de Pascua
    - NewYear: Contiene la estrategia para la temporada de Año Nuevo

Problema No. 3

Contexto:

Se ocupa crear una funcionalidad de trabaje con el esquema jerárquico de puestos de una compañía (con n niveles o áreas / subáreas) y cantidad variable de empleados en

cada una de ellas de modo que se pueda realizar un listado secuencial de los empleados que pertenecen a esta organización. Como verá, algunos de las piezas en el organigrama representan áreas (nodos internos de la estructura jerárquica) y otras personas dentro de ellas. Nos interesa la información de las personas únicamente (es decir los “nodos” hoja) y obtenerla de manera secuencial.

Patrón Propuesto: ( ) Comportamiento
Iterator

Diagrama original del patrón:

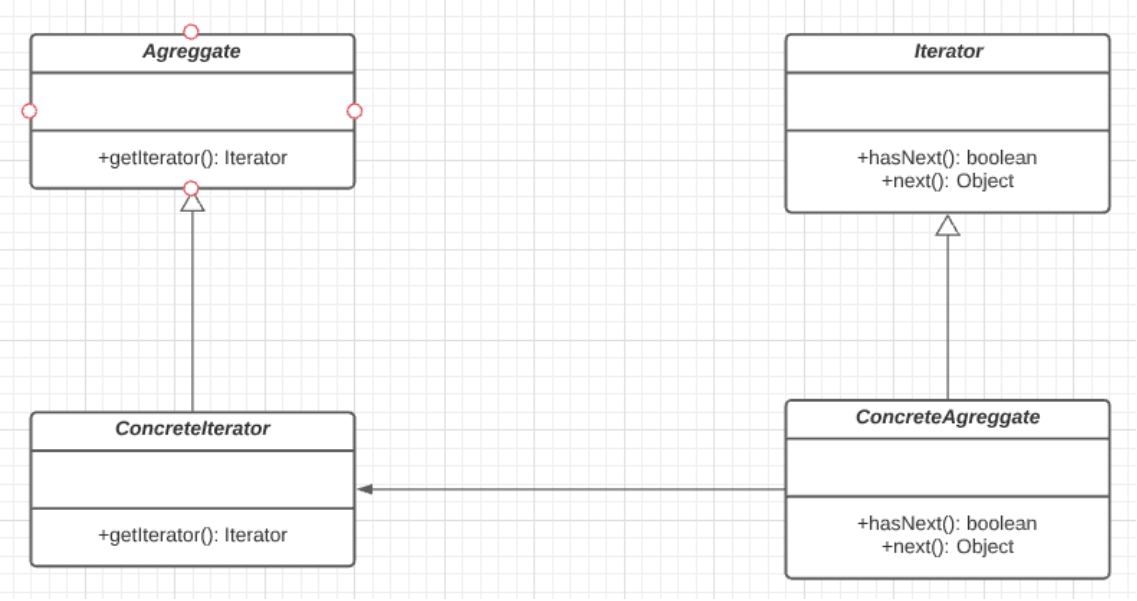
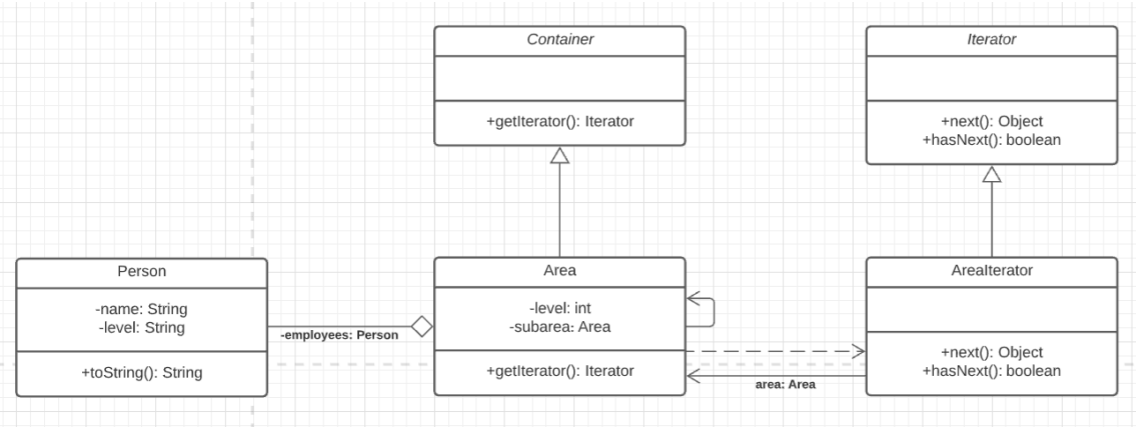


Diagrama del patrón en el contexto del problema:



Rol de cada elemento del patrón en la solución propuesta:

- Container: Interface que tiene método para obtener un Iterador

- Area: Clase concreta que implementa este método de obtener el iterador concreto, además de tener un número de nivel jerárquico, una lista de empleados y una subárea correspondiente
- Person: Clase que representa al empleado que puede estar en un área de la compañía
- Iterator: Interface que tiene método next y hasNext para iterar la estructura de la compañía
- AreaIterator: Clase concreta que implementa método que itera sobre la estructura de la compañía.

Problema No. 4

Contexto: Se necesita construir una aplicación que tome comandos SQL para realizar consultas sobre una Archivo de Excel, como si este se tratara de una base de datos relacional, en donde cada Hoja será vista como una tabla y las columnas de la hoja como columnas de la tabla. Se requiere contar con una estructura de clases que represente el lenguaje SQL para que a través de ellas se pueda producir el resultado.

Patrón Propuesto: ( ) Comportamiento
Interpreter

Diagrama original del patrón:

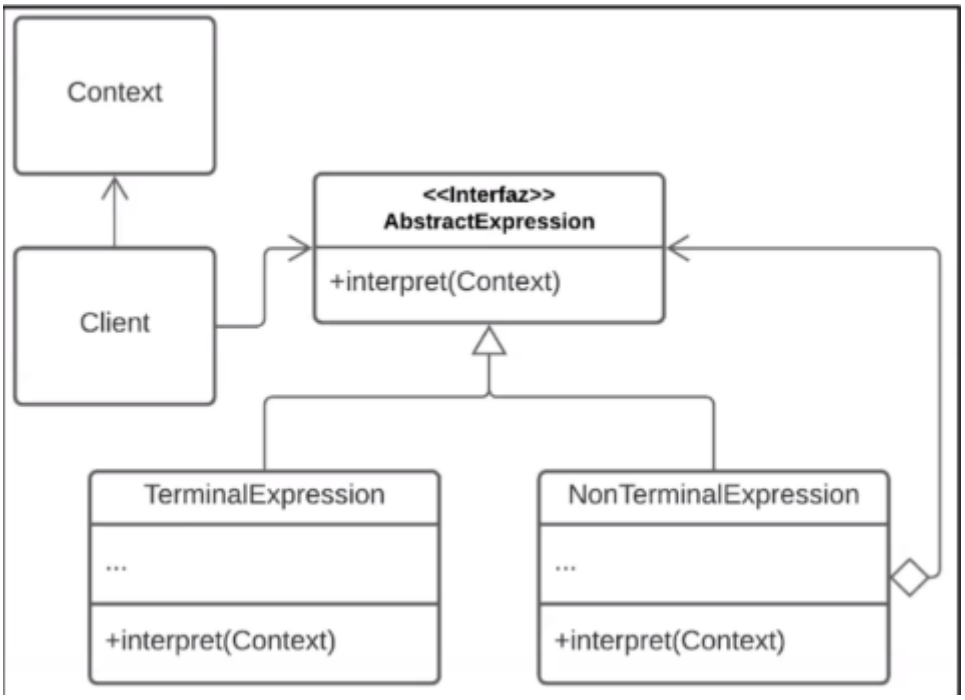
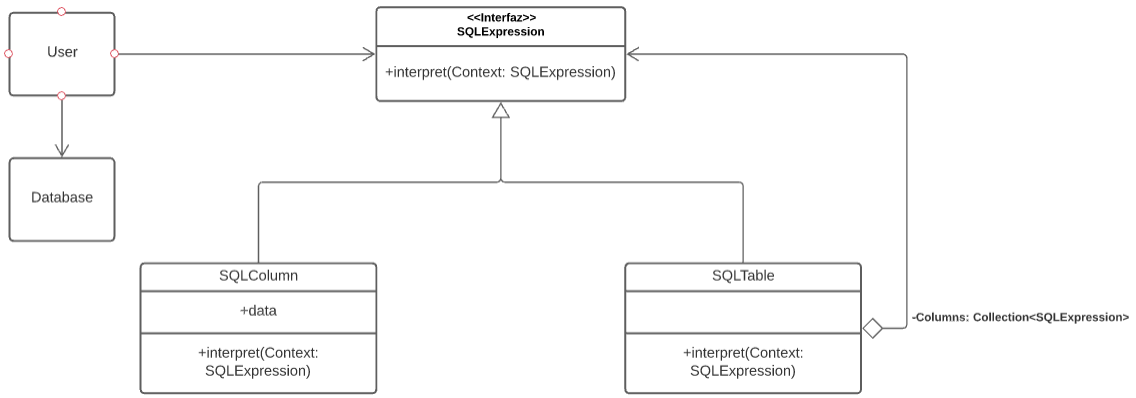


Diagrama del patrón en el contexto del problema:



Rol de cada elemento del patrón en la solución propuesta:

- Database: Almacena la estructura de la SQLExpression y contiene la información global de las tablas.
- User: Es el usuario que ejecuta la expresión de SQL de consulta a las tablas.
- SQLExpression: Es el interface que representa los queries de la base de datos.
- SQLColumn: Representa la columna de una tabla de la base de datos.
- SQLTable: Representa una tabla de una base de datos. Puede tener otras tablas contenidas como columnas.

Problema No. 5

Contexto:

En una biblioteca se maneja un inventario de libros de los que se lleva la existencia por título, autor y edición.

Cada vez que se realiza un préstamo o devolución de un ejemplar, se debe actualizar el estado del inventario, para que quienes estén consultando disponibilidades conozcan las existencias reales. Interesa aportar las vistas para poder consultar las existencias de un ejemplar y tramitar un préstamo / devolución.

Patrón Propuesto: ( ) Comportamiento

Observer

Diagrama original del patrón:

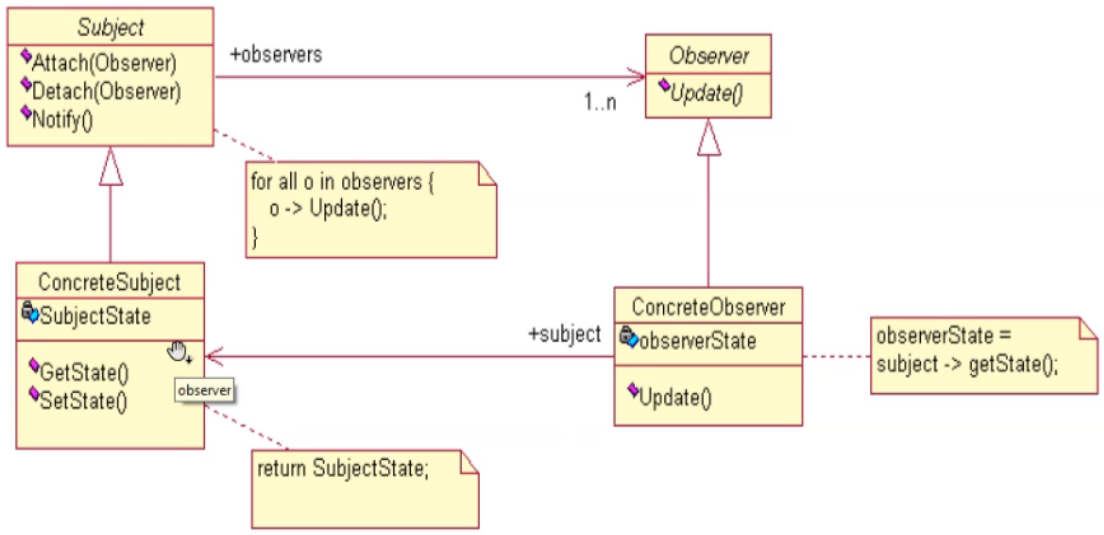
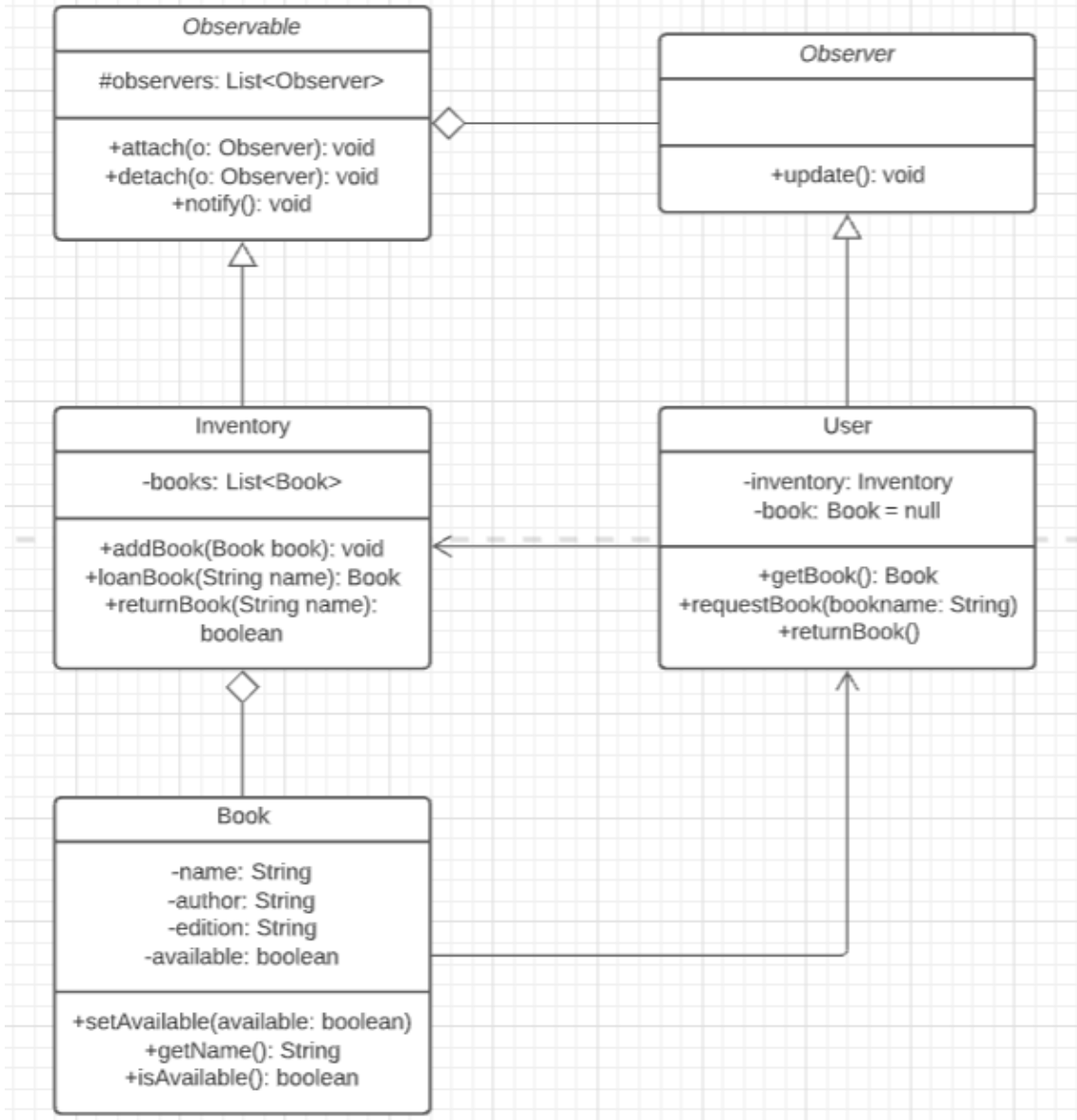


Diagrama del patrón en el contexto del problema:



Rol de cada elemento del patrón en la solución propuesta:

- Observable: Clase Abstracta que tiene lista de observadores y métodos abstractos de agregarlos o eliminarlos, así como notificar a cada uno de estos.
- Inventory: Clase concreta que implementa métodos de Observable y tiene lista de libros existentes y métodos para alquilar y retornar alguno de estos.
- Book: Clase que tiene información de un libro del inventario
- Observer: Clase abstracta que tiene método abstracto de actualizar el sujeto observado que tiene
- User: Clase concreta que implementa la actualización de observador y puede alquilar o devolver un libro del inventario

Estas fichas deben quedar subidas en el muro de publicaciones del canal privado al final de la clase asincrónica como evidencia de su trabajo de clase.

Segunda parte. Propuesta de un contexto para un patrón asignado

Adicionalmente, cada equipo deberá redactar un contexto de problema que pueda resolverse con un patrón asignado de acuerdo a la siguiente tabla:

Equipo	Patrón Asignado		Equipo	Patrón Asignado
01	State		06	Intérprete
02	Observer		07	State
03	Iterator		08	Observer
04	Intérprete		09	Strategy
05	Strategy		10	Iterator

Coloque su contexto y su solución propuesta en UML en el siguiente cuadro:

Patrón de Comportamiento para redacción de un contexto: Interpreter
Contexto Propuesto: Se necesita construir un programa que lea expresiones algebraicas en inglés para obtener su resultado numérico. Este programa lee expresiones de suma o resta de números del 0-9.
Diagrama original del patrón:

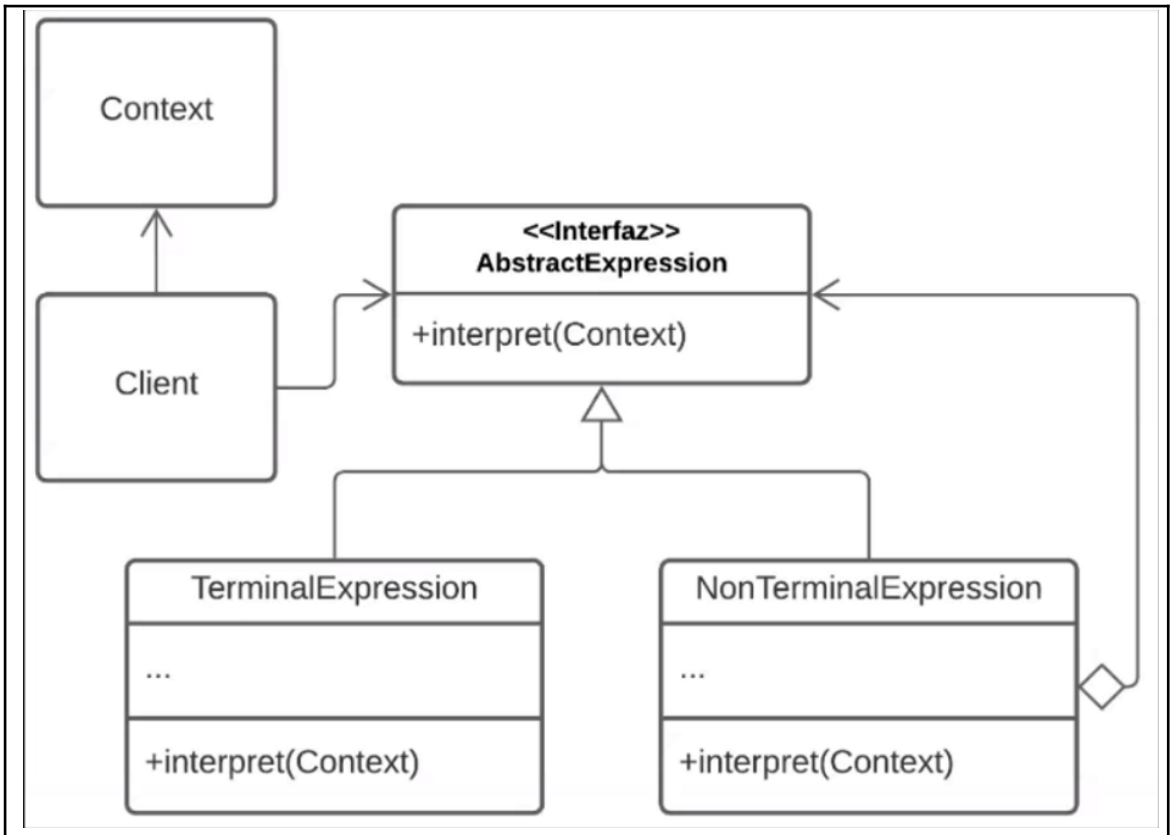
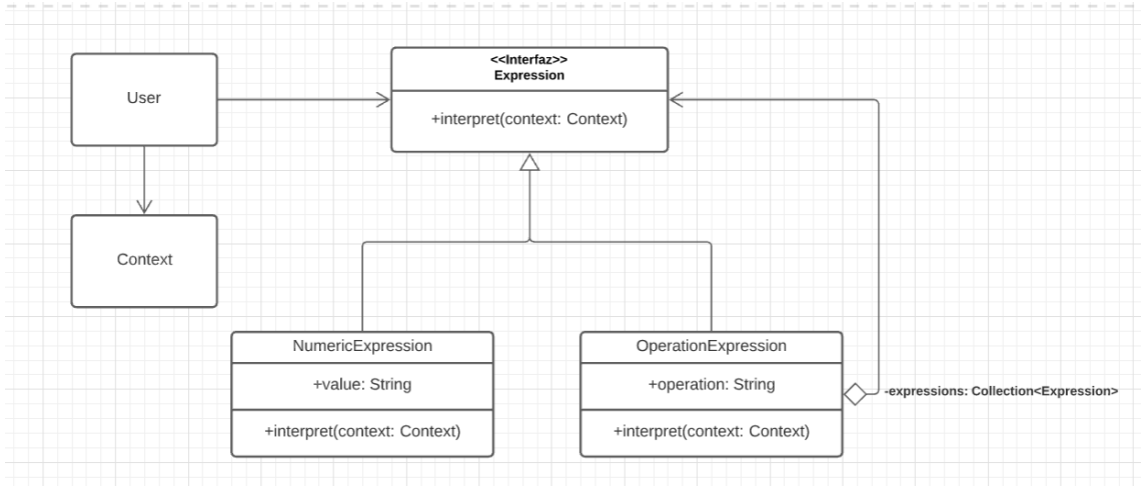


Diagrama del patrón en el contexto del problema:



Rol de cada elemento del patrón en la solución propuesta:

- Expression: Representa la expresion abstracta, la cual tiene de clases hijas las numéricas y las de operacion.
- NumericExpression: Representa una expresión numérica terminal. Estas tienen un valor en string que es interpretado por el contexto.
- OperationExpression: Representa una expresión de operación no terminal. Estas tienen un valor en string que es interpretado por el contexto.

- User: Representa a la clase auditora que hace el llamado a interpretar la expresión y obtener el resultado.
- Context: Representa el contexto que interpreta los strings de las expresiones que se reciban en el programa.

Tercera parte. Programación de casos

De los casos de la primera parte se asigna un caso para que el equipo programe su propuesta de acuerdo a lo definido en la primera parte.

Casos	1	2	3	4	5
Equipo a cargo	04-09	03-10	02-07	05-08	01-06

Para efectos de comprender entonces la instrucción se pone el siguiente ejemplo:

El equipo 01 completa todas las fichas de los 5 casos durante la sesión asincrónica del martes 25 de mayo. Posteriormente propone un contexto de acuerdo al patrón STATE asignado en la segunda parte y programa en la tercera parte tanto el contexto propuesto de la segunda parte como la solución del caso 5.

En esta práctica se abarcarán 5 patrones de compartimiento de los 11 correspondientes. Los otros 6 serán evaluados en una segunda práctica final luego de las exposiciones pendientes.

Entregables:

1. Imágenes de las tablas de resolución de casos en muro de publicaciones del canal privado de cada equipo antes de mediodía del martes 25 de mayo.

2. Se entrega este documento completo con el proyecto con los dos casos programados para el jueves 03 de junio al final del día

Caso 1 (STATE)

Se debe simular el comportamiento de un servidor que atiende peticiones de manera concurrente. En ciertas ocasiones, como cuando se toma un cierto periodo de descanso, se planea apagarlo por completo, cuya acción a manera interna genera una teminación y desconexión de los servicios activos, se crean uso registros de log y el servidor se apaga de manera controlada. Cuando se retorna del descanso, debe iniciarse de nuevo y esto toma algunos segundos porque debe llevar a cabo el arranque de varios servicios y realizar algunas inicializaciones.

Una vez “levantado” el servidor puede iniciar su proceso de atención de peticiones que provienen de distintos clientes, las cuales son atendida en un tiempo promedio. No obstante, en momentos, poco comunes, el servidor puede entrar en un modo de saturación debido a la cantidad excesiva de peticiones, en este caso, “rechaza” las peticiones entrantes hasta que haya alcanzado un nivel adecuado de manejo y luego vuelve a restablecer la posibilidad de atención.

En cualquier momento, en caso de que ocurra algo inesperado a nivel del entorno físico del mismo, el servidor podría apagarse automáticamente.

Caso 2 (STRATEGY)

Una tienda por departamentos ha decido realizar una campaña de descuentos en productos y la mantendrá durante el año aplicando un porcentaje de descuento específico dependiendo de la temporada del año: Navidad, Easter, Año Nuevo. Se desea que el sistema de cobro aplique el descuento establecido de manera automática sin que el dependiente en cajas deba anotar el porcentaje que se ha fijado. La tienda valorará introducir otras épocas o temporadas para agregar nuevos descuentos, como la entrada de verano o invierno.

Caso 3 (ITERATOR)

Se ocupa crear una funcionalidad de trabaje con el esquema jerárquico de puestos de una compañía (con n niveles o áreas / subáreas) y cantidad variable de empleados en

cada una de ellas de modo que se pueda realizar un listado secuencial de los empleados que pertenecen a esta organización. Como verá, algunos de las piezas en el organigrama representan áreas (nodos internos de la estructura jerárquica) y otras personas dentro de ellas. Nos interesa la información de las personas únicamente (es decir los “nodos” hoja) y obtenerla de manera secuencial.

Caso 4 (INTERPRETER)

Se necesita construir una aplicación que tome comandos SQL para realizar consultas sobre una Archivo de Excel, como si este se tratara de una base de datos relacional, en donde cada Hoja será vista como una tabla y las columnas de la hoja como columnas de la tabla. Se requiere contar con una estructura de clases que represente el lenguaje SQL para que a través de ellas se pueda producir el resultado.

Caso 5 (OBSERVER)

En una biblioteca se maneja un inventario de libros de los que se lleva la existencia por título, autor y edición.

Cada vez que se realiza un préstamo o devolución de un ejemplar, se debe actualizar el estado del inventario, para que quienes estén consultando disponibilidades conozcan las existencias reales. Interesa aportar las vistas para poder consultar las existencias de un ejemplar y tramitar un préstamo / devolución.