PROGRAMACIÓN ORIENTADA A OBJETOS

Trabajo Práctico Especial 2018

Integrantes

- **Nombre y Apellido:** Leandro Salias
  - **Email:** leandrosalias42@gmail.com

- **Nombre y Apellido:** Gonzalo Clementi
  - **Email:** gonzaloclementi@gmail.com

Introducción

Este informe se centra en la explicación sobre cómo fue la realización y decisiones de implementación del Trabajo Práctico Especial de la materia, del corriente año. Dicho trabajo se basó en la realización de una estructura de almacenamiento de múltiples colas de objetos con múltiples niveles. La misma debe permitir agregar o recuperar los objetos que se guarden en esta.

El trabajo consta de 3 incisos, donde en cada uno de ellos se podrá aplicar todos los mecanismos aprendidos de la programación orientada a objetos.

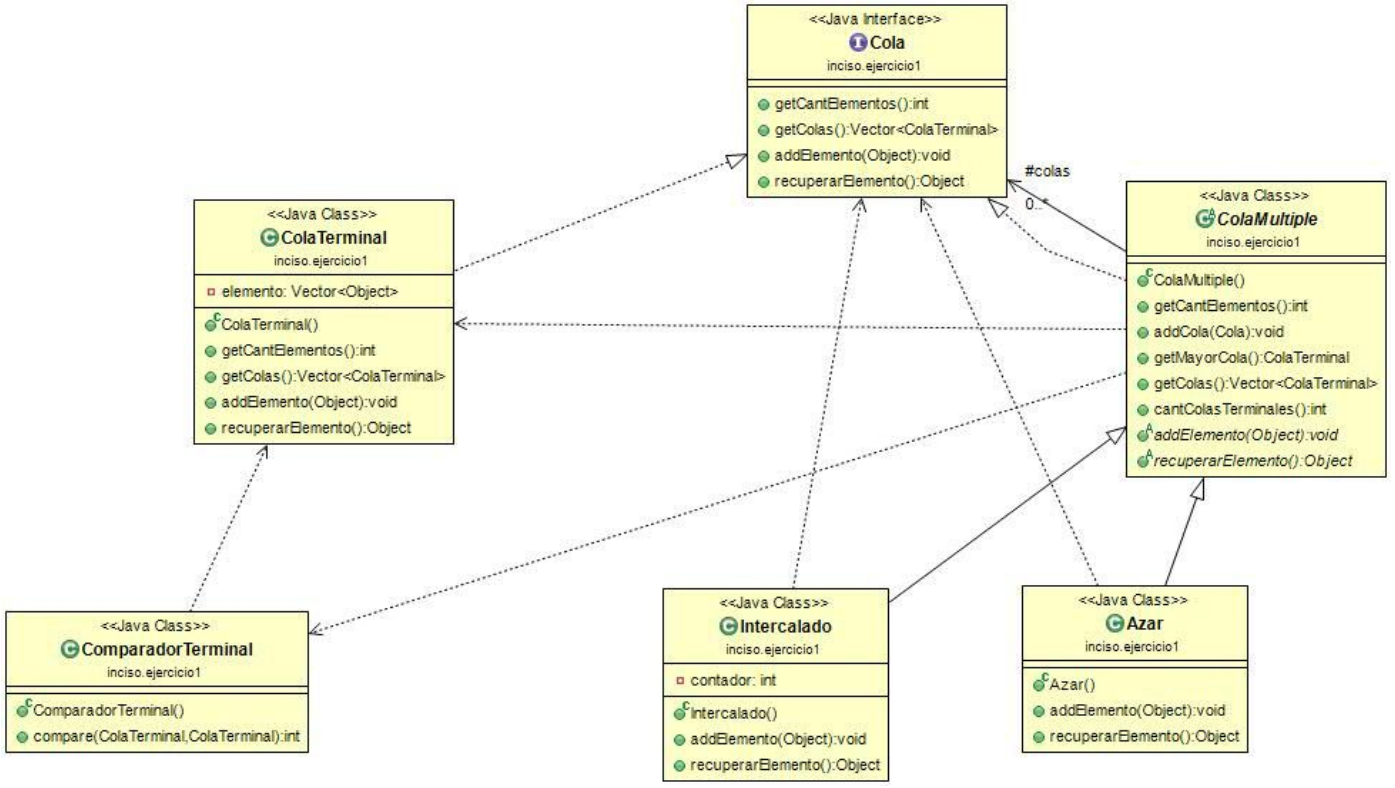
Inciso 1

Para el inciso 1 se nos requería implementar una estructura de almacenamiento de múltiples colas, las cuales están compuestas por otras colas múltiples que agregan y recuperan al azar o intercalado (sólo de una forma) y por colas terminales que funcionan como una cola común (agregan al final, recuperan del inicio). Además no se nos permitía variar la forma de agregar y recuperar en tiempo de ejecución, es decir, que debía ser definida de manera estática.
Identificamos dos tipos de colas, la cola múltiple y la cola terminal, por lo que decidimos abstraer las funcionalidades en común (agregar, recuperar, obtener la cantidad de elementos y devolver la cola) en una interfaz Cola. Esta interfaz va a ser implementada tanto por ColaTerminal como por ColaMultiple, cada una de éstas a su manera, ya que el comportamiento de sus métodos en común varían.
Como la clase ColaMultiple tiene dos formas de agregar y recuperar, definimos estos métodos como abstractos, transformando esta clase en una clase abstracta, ya que la funcionalidad de estos métodos serán implementados en las clases Azar e Intercalado (con sus correspondientes comportamientos).
Además de las funcionalidades básicas de una cola como lo son agregar y recuperar se solicitaron otras operaciones. Una de ellas requería devolver la cola terminal con más elementos, por lo que decidimos implementar un comparador que compara colas terminales por cantidad de elementos que contienen. Esto para luego poder utilizar el método de la clase Collection, Collection.max(), que utiliza el comparador. El método compare(ColaTerminal c1, ColaTerminal c2) se implementa en una clase ComparadorTerminal que implementa Comparator.

Otras operaciones que se solicitaron son calcular la cantidad total de elementos de la estructura y determinar la cantidad de colas terminales que contiene una cola, ambas implementadas en ColaMultiple.

Para la primera recorremos con un for desde el principio de la ColaMultiple y a medida que recorre va sumando la cantidad de elementos que contienen las colas que la componen; con el polimorfismo se van a llamar tanto el getCantElementos de las colas múltiples como el de las colas terminales, para finalmente retornar la suma total. Para la cantidad de colas terminales que contiene una cola utilizamos el método getColas que devuelve un vector con todas las colas terminales que contiene una cola múltiple, para luego en cantColasTerminales retornar el tamaño del vector.

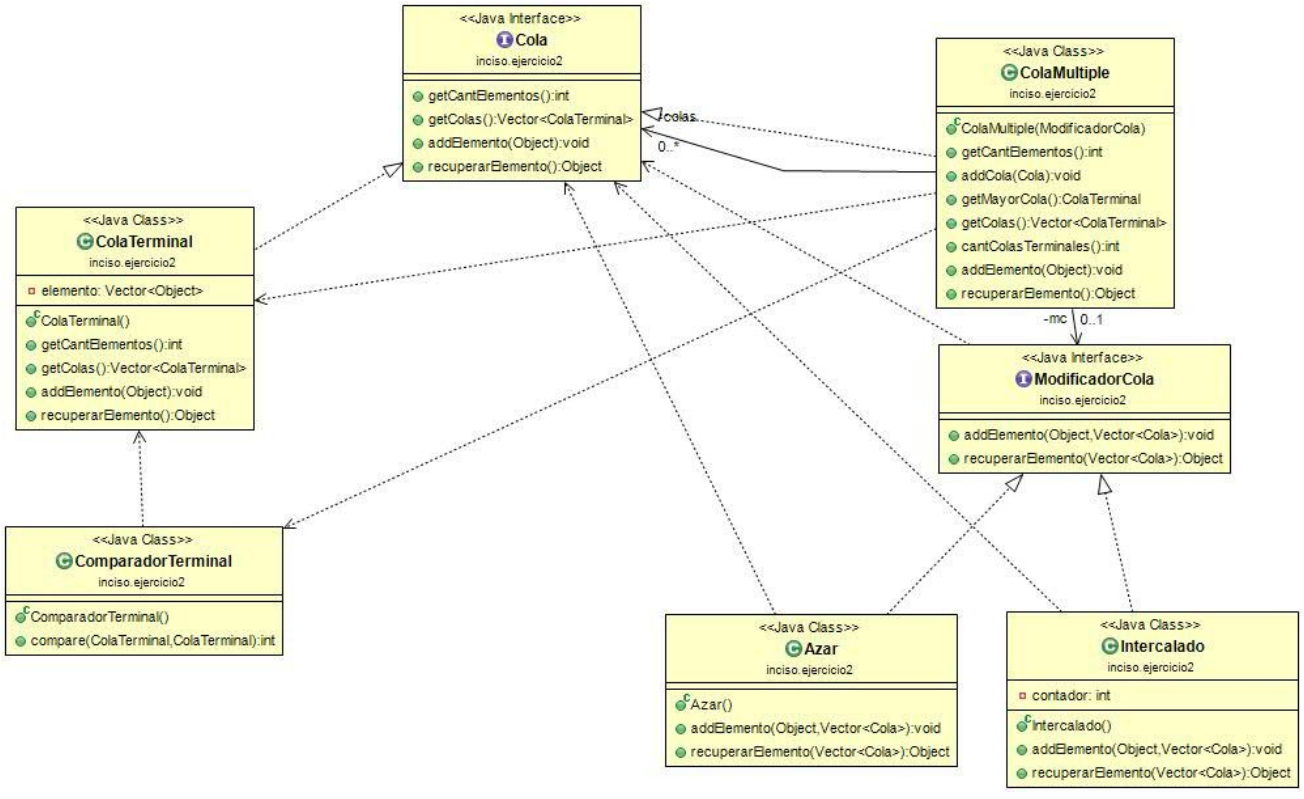
El diagrama de clases para este inciso es el siguiente:



Inciso 2

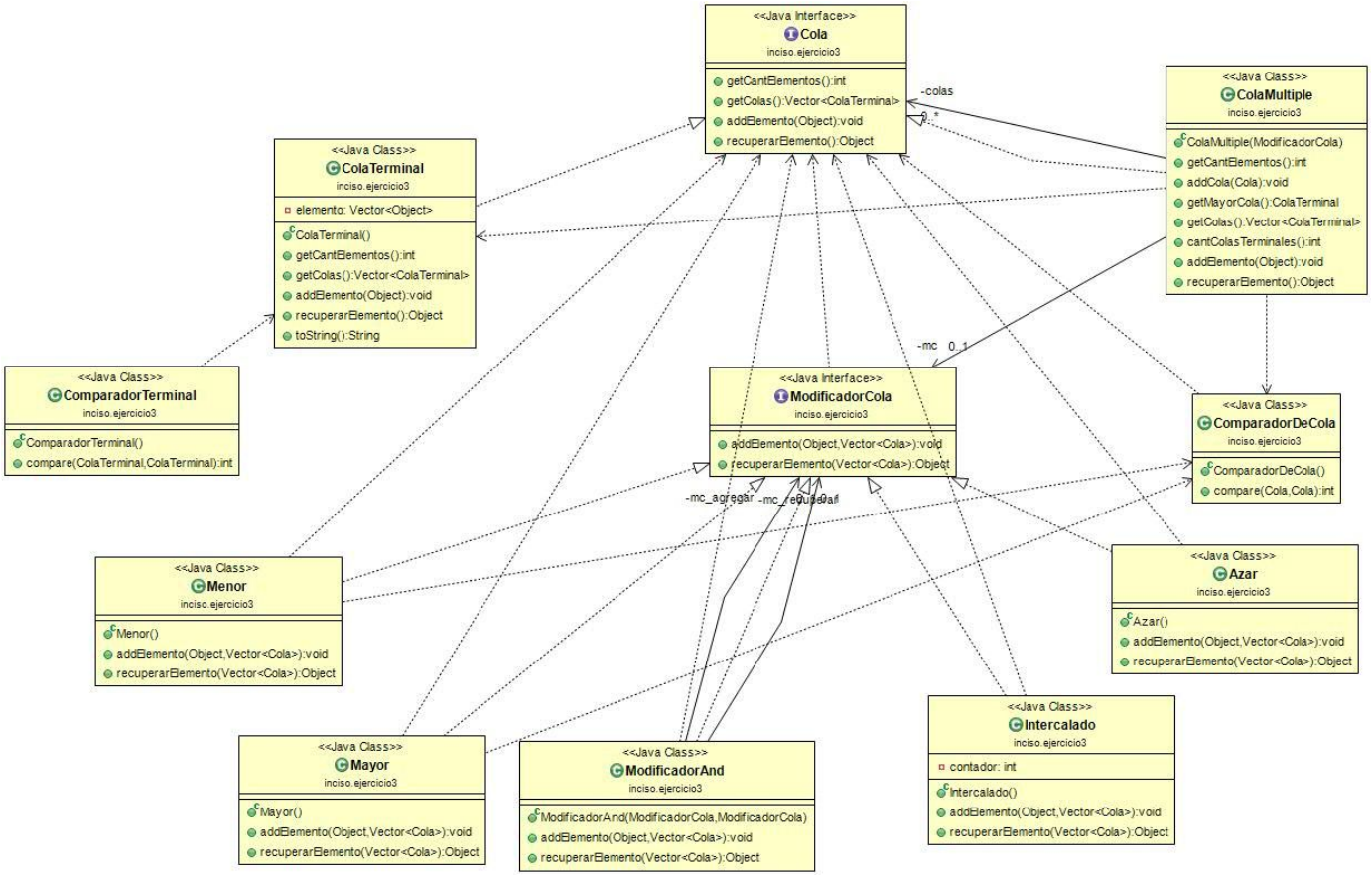
Para el inciso 2, se nos pide con la resolución del punto anterior (que fue resuelto de manera estática), que la forma de agregar y recuperar sea de manera dinámica, es decir, que se pueda variar en tiempo de ejecución.
Para ello, creamos una nueva interfaz llamada ModificadorCola, que esta misma contiene los métodos addElemento y recuperarElemento, (que son métodos abstractos) y las clases Azar e Intercalado implementan esos métodos con su respectivo comportamiento. Esta nueva interfaz, permite poder variar la manera en que se agregan y recuperan los elementos (ya sea al azar o intercalado) en tiempo de ejecución del programa, lo que hace que la solución sea dinámica. Esta interfaz también es utilizada por la clase ColaMultiple, ésta última teniendo un atributo del tipo ModificadorCola, y en su constructor se le pasa por parámetro otro objeto ModificadorCola, ya que esto permite construir una ColaMultiple que agregue y recupere de manera al Azar o Intercalado.
Otra diferencia con respecto al inciso anterior, es que la clase ColaMultiple que era abstracta, pasó a ser una clase concreta, ya que agregamos la interfaz ModificadorCola y ColaMultiple, deja de tener la responsabilidad de tener los métodos de agregar y recuperar de manera abstracta.

El diagrama de clases para este inciso es el siguiente:



Inciso 3

Para el último ejercicio se nos solicitó agregar a la implementación del inciso 2, dos nuevas formas de agregar y recuperar, que son recuperar y agregar de la cola más grande y recuperar y agregar de la cola más pequeña. Además, las formas de agregar y recuperar pueden combinarse, es decir, se puede agregar al azar y recuperar de la más grande en una misma cola.
Decidimos no modificar la estructura de la ColaMultiple y para esto creamos una nueva clase llamada ModificadorAnd. Ésta tiene como atributos dos modificadores de cola de los cuáles uno se utiliza exclusivamente para agregar y otro solamente para recuperar. Luego para combinar cualquier forma de agregar y recuperar, se creará una instancia de ModificadorAnd a la que se le pasará como parámetros dos ModificadorCola de cualquier tipo (Azar, Intercalado, Menor o Mayor) y luego este ModificadorAnd se pasará como parámetro al construir una ColaMultiple.
Además creamos dos nuevas clases, la clase Mayor y la Menor, que implementarán la interfaz ModificadorCola. La primera tiene como función recuperar y agregar de la Cola con mayor cantidad de elementos, y la segunda la tarea de agregar y recuperar de la Cola con menor cantidad de elementos. Tanto para los recuperar y los agregar de estas dos nuevas clases utilizamos el ComparadorCola, que es el ComparadorTerminal modificado para que compare objetos de tipo Cola y no solamente ColaTerminal. El diagrama de clases para este inciso es el siguiente:



Conclusión

Como conclusión, a pesar de haber tenido algunas dificultades para entender el enunciado, las dudas fueron despejadas y logramos implementar la solución de la manera que nos pareció más sencilla, descartando algunas ideas que tal vez eran mucho más complicadas y que no utilizaban tanto los pilares de la programación orientada a objetos.
Con la realización de este trabajo, se nos permitió conocer e interpretar todos los mecanismos de la programación orientada a objetos, como la abstracción, binding dinámico, polimorfismo, herencia, encapsulamiento, aplicación de interfaces y la distribución de prioridades al momento de realizar el modelado e implementación de un programa.