Proyecto #1: Colecciones en Java
Estructura de datos y algoritmos II

López, Ricardo Argüello, Dante Sánchez, Marco

5 de noviembre de 2020

1. Objetivo

Que el alumno conozca los principales aspectos teórios y prácticos de las colecciones y sus aplicaciones en el lenguaje de programación Java, así mismo que ponga en práctica los conceptos básicos de la programación orientada a objetos y el trabajo en equipo a distancia.

2. Marco Teórico

2.1. ¿Qué es una interfaz?

Una interfaz es una lista de métodos (solamente cabeceras de métodos, sin implementación) que define un comportamiento específico para un conjunto de objetos. Cualquier clase que declare que implementa una determinada interfaz, debe comprometerse a implementar todos y cada uno de los métodos que ese interfaz define. Esa clase, además, pasará a pertenecer a un nuevo tipo de dato extra que es el tipo de la interfaz que implementa. Los interfaces actúan, por tanto, como tipos de clases. Se pueden declarar variables que pertenezcan al tipo de la interfaz, se pueden declarar métodos cuyos argumentos sean del tipo de la interfaz, asimismo se pueden declarar métodos cuyo retorno sea el tipo de una interfaz.

Una interfaz es un conjunto de constantes y de métodos abstractos.

2.1.1. Clases

Cuando se define una clase, se especifica como serán los objetos de dicha clase, esto quiere decir, de que variables y de que métodos estará constituida. Las clases proporciona una especie de plantilla o molde para los objetos van a tener como atributos (características), y como acciones o métodos(acciones).

2.2. Hashing

El hashing es un concepto muy importante en computación ya que permite recuperar elementos de una colección con una complejidad temporal O(1). Para acceder a cualquier localización de una lista debemos saber su índice. Un índice sirve como la dirección de un elemento una vez que ha sido almacenado en la lista. Para lograr acceso a un índice en tiempo O(1), debemos recordar el índice del elemento almacenado, de otra manera el elemento se tendrá que buscar, operación con complejidad temporal O(n), no O(1).

Entonces, si queremos recuperar los elementos de una lista en tiempo O(1), de alguna manera hay que recordar los índices de todos los elementos. A primera vista esto parece imposible, pero utilizando a los mismos elementos a almacenar como base para encontrar la dirección, se pueden recuperar elementos en O(1). Entonces, a través de funciones que transforman alguna característica del elemento a almacenar, se calculan las direcciones de los elementos. Este es el principio del hashing que permite que se puedan recuperar datos en O(1).

Estructuras de datos como los mapas y los conjuntos aplican este concepto para almacenar y recuperar elementos.

2.3. Collection Framework de Java

La plataforma Java incluye un marco de colecciones (Collection Framework). Un marco de colecciones es una arquitectura unificada para representar y manipular colecciones, lo que permite manipular colecciones independientemente de los detalles de implementación.

3. Colecciones

Una colección representa un grupo de objetos. Algunas colecciones permiten elementos duplicados y otras no. Algunos están ordenados y otros desordenados. El JDK no proporciona ninguna implementación directa de esta interfaz: proporciona implementaciones de subinterfaces más específicas como Set y List. Esta interfaz se utiliza normalmente para pasar colecciones y manipularlas donde se desea la máxima generalidad.

3.1. Iterable

Un Iterable es una interface que hace referencia a una colección de elementos que se puede recorrer.

La interface solo necesita que implementemos un método para poder funcionar de forma correcta, este método es iterator().

Esto significa que una clase que implementa la interfaz Java Iterable puede tener sus elementos iterados.

3.2. Interfaz Collection

La interfaz raíz en la jerarquía de colecciones.Todas las clases de implementación de Collection de propósito general (que normalmente implementan Collection indirectamente a través de una de sus subinterfaces) deben proporcionar dos constructores .estándar": un constructor vacío (sin argumentos), que crea una colección vacía, y un constructor con un solo argumento de tipo Colección, que crea una nueva colección con los mismos elementos que su argumento.

Esta interfaz es miembro de Java Collections Framework.

4. Conjuntos

4.1. Interfaz Set

Una colección que no contiene elementos duplicados. Más formalmente, los conjuntos no contienen un par de elementos e1 y e2 tales que e1.equals (e2), y como máximo un elemento nulo. Como lo implica su nombre, esta interfaz modela la abstracción de conjuntos matemáticos.

Todos los constructores deben crear un conjunto que no contenga elementos duplicados (como se definió anteriormente). No está permitido que un conjunto se contenga a sí mismo como un elemento. Algunas implementaciones de conjuntos tienen restricciones sobre los elementos que pueden contener. Por ejemplo, algunas implementaciones prohíben los elementos nulos y algunas tienen restricciones sobre los tipos de sus elementos.

Esta interfaz es miembro de Java Collections Framework.

4.1.1. Clase HashSet

Esta clase implementa la interfaz Set, respaldada por una tabla hash (en realidad, una instancia de HashMap). No ofrece ninguna garantía en cuanto al orden de iteración del conjunto; en particular, no garantiza que el pedido se mantenga constante en el tiempo. Esta clase permite el elemento nulo.

Esta clase ofrece un rendimiento de tiempo constante para las operaciones básicas (add, remove, contains and size), asumiendo que la función

hash dispersa los elementos correctamente entre los depósitos. La iteración sobre este conjunto requiere un tiempo proporcional a la suma del tamaño de la instancia de HashSet (el número de elementos) más la çapacidad"de la instancia de HashMap de respaldo (el número de depósitos). Por lo tanto, es muy importante no establecer la capacidad inicial demasiado alta (o el factor de carga demasiado bajo) si el rendimiento de la iteración es importante.

Esta clase es miembro de Java Collections Framework.

4.1.2. Clase LinkedHashSet

Implementación de tabla hash y lista vinculada de la interfaz Set, con orden de iteración predecible. Esta implementación se diferencia de HashSet en que mantiene una lista doblemente enlazada que se ejecuta en todas sus entradas. Esta lista enlazada define el orden de iteración, que es el orden en el que se insertaron los elementos en el conjunto (orden de inserción). Se puede utilizar para producir una copia de un conjunto que tiene el mismo orden que el original, independientemente de la implementación del conjunto original.

Esta clase proporciona todas las operaciones Set opcionales y permite elementos nulos. Al igual que HashSet, proporciona un rendimiento en tiempo constante para las operaciones básicas (add, contains and remove), asumiendo que la función hash dispersa los elementos correctamente entre los depósitos. Es probable que el rendimiento sea ligeramente inferior al de HashSet, debido al gasto adicional de mantener la lista vinculada.

Esta clase es miembro de Java Collections Framework.

4.2. Interfaz SortedSet

Un conjunto que además proporciona un ordenamiento total de sus elementos. Los elementos se ordenan utilizando su orden natural o mediante un comparador que normalmente se proporciona en el momento de la creación del conjunto ordenado. El iterador del conjunto atravesará el conjunto en orden ascendente de elementos. Se proporcionan varias operaciones adicionales para aprovechar el pedido. (Esta interfaz es el análogo establecido de SortedMap).

Todos los elementos insertados en un conjunto ordenado deben implementar la interfaz Comparable (o ser aceptados por el comparador especificado). Además, todos estos elementos deben ser mutuamente comparables.

Todas las clases de implementación de conjuntos ordenados de propósito general deben proporcionar cuatro constructores .estándar": 1) Un construc-

tor vacío (sin argumentos), que crea un conjunto ordenado vacío ordenado según el orden natural de sus elementos. 2) Un constructor con un solo argumento de tipo Comparator, que crea un conjunto ordenado vacío ordenado según el comparador especificado. 3) Un constructor con un solo argumento de tipo Colección, que crea un nuevo conjunto ordenado con los mismos elementos que su argumento, ordenados según el orden natural de los elementos. 4) Un constructor con un solo argumento de tipo SortedSet, que crea un nuevo conjunto ordenado con los mismos elementos y el mismo orden que el conjunto ordenado de entrada.

Esta interfaz es miembro de Java Collections Framework.

5. Listas

5.1. Interfaz List

La interfaz List, también conocida como "secuencia" es la implementación de la estructura de datos List, y normalmente acepta elementos repetidos o duplicados según sus implementaciones, y donde cada uno de los elementos se encuentra indexado, el primer elemento se encuentra en la posición cero. Hereda todos los métodos de la interfaz Collection.

Esta interfaz proporciona el uso de un iterador especial llamado ListIterator que extiende a la interfaz Iterator. La interfaz List incluye métodos para acceder posicionalmente a los elementos (por medio del índice), para realizar búsqueda secuencial, y hacer u obtener operaciones con sublistas, En Java existen dos implementaciones muy importantes de la interfaz List: ArrayList y LinkedList.

5.1.1. Clase ArrayList

ArrayList es una de las implementaciones más usadas de la Interfaz List, pues representa el uso de un Arreglo Dinámico que puede modificar su tamaño en tiempo de ejecución. Esta clase incluye métodos para manipular el tamaño del arreglo que internamente contiene los elementos de una List: size, isEmpty, get, set, add, Iterator y ListIterator.

Cada objeto de tipo ArrayList tiene una capacidad, que es el tamaño del arreglo usado para almacenar los elementos de la lista, siempre de que se agregue un nuevo elemento se aumentará la capacidad automáticamente. Es importante saber que esta implementación no está sincronizada, por lo que no es seguro manipular un ArrayList con varios subprocesos a la vez a menos que se sincronice de manera externa.

Esta implementación es preferida ante un Array convencional dado su dinamismo, sin embargo, la clase Vector es equivalente a un ArrayList, excepto que el Vector sí es seguro para manejar hilos.

5.1.2. Clase LinkedList

La clase LinkedList representa una implementación de la estructura de datos lineal lista ligada, donde cada uno de los nodos contiene una referencia a su nodo contiguo, esto no significa que estén contiguos físicamente en memoria, pueden estar separados pero enlazados por medio de referencias. Esta es otra de las implementaciones más usadas de la Interfaz List, sin embargo, tienen algunas desventajas con respecto a ArrayList, pues LinkedList toma mayor cantidad de tiempo para encontrar un elemento ya que debe recorrer todos siguiendo la secuencia indicada por las referencias hasta hallar el valor buscado, y ArrayList sólo necesita saber el índice del elemento y accede directamente a él.

La clase LinkedList contiene todos los métodos que tiene un ArrayList, pero su principal diferencia radica en cómo están construidas: Mientras que ArrayList en esencia tiene un arreglo que contiene sus elementos, un LinkedList almacena sus elementos en nodos referenciados. Los nuevos métodos que contiene son: addFirst, addLast, removeFirst, removeLast, getFirst y getLast.

5.1.3. Clase Vector

Un Vector es en esencia igual a un ArrayList, pues ambas implementaciones tienen un Array internamente como estructura de datos, ambos pueden modificar el tamaño del arreglo de manera dinámica, pero su diferencia es cómo modifican su capacidad. Por defecto un Vector duplica el tamaño de su arreglo cuando un elemento se agrega, mientras que ArrayList aumenta el tamaño agregando la mitad de su tamaño actual por cada elemento ingresado. Además de que Vector fue incluido en la primera versión del JDK, mientras que ArrayList fue incluido en la versión 1.2.

Pero su principal diferencia es que un Vector es sincronizado, esto implica que puede ser modificado seguramente por varios subprocesos, mientras que ArrayList no. Que un Vector sea sincronizado significa un mayor coste de recursos, y sólo lo debemos usar cuando tengamos que hacer operaciones “thread-safe”.

5.1.4. Clase Stack

La clase Stack extiende a la clase Vector para implementar una estructura de datos de tipo pila. Una pila es una estructura de tipo LIFO, donde el último elemento ingresado es el primero que sale, e incluye los métodos push (apilar), pop (desapilar), peek (copiar el elemento de la cima), search (busca la posición de un elemento) y empty (verificar si es vacía). Dado que incluye todos los métodos de Vector, podemos usar el método add para añadir en cualquier posición, sin embargo, no es recomendable puesto que se perdería la naturaleza de la Pila.

Un conjunto más completo y consistente para operaciones de Pila es el de la interfaz Deque y sus implementaciones.

6. Colas

6.1. Interfaz Queue

Es una colección diseñada para almacenar elementos antes de ser procesados. Esta estructura generalmente pero no necesariamente ordena los elementos de manera FIFO (first-in-first-out). Una excepción de esta esto son las colas LIFO o stacks. La mayor parte de las implementaciones no tienen un límite de elementos, pero algunas otras sí.

Las implementaciones de queue generalmente no permiten elementos nulos ya que muchos métodos devuelven null al realizar sus operaciones.

Algunos de los métodos más importantes de esta interfaz son:

- Add y offer permiten añadir elementos a una cola si es posible. El primero devuelve una excepción si no hay espacio disponible y el segundo devuelve verdadero o falso dependiendo de si se pudo añadir un elemento o no.
- Peek y element devuelven, pero no remueven, el primero elemento de una cola. El primero devuelve null si la cola está vacía y el segundo devuelve una excepción.
- Poll y remove devuelven y eliminan al primer elemento de una cola. El primero devuelve null si la cola está vacía y el segundo genera una excepción.

6.1.1. Clase priorityQueue

Esta es una implementación de una cola de prioridad basada en un heap de prioridad. En esta estructura, los elementos están ordenados de acuerdo con su orden natural o por un comparador provisto durante la creación de la estructura. Esta estructura no permite la inserción de elementos no comparables si se está ordenando a los elementos de acuerdo con su orden natural.

Esta implementación no está sincronizada, lo que significa que diferentes procesos pueden acceder a ella al mismo tiempo.

Además, esta clase ofrece complejidad O(log(n)) para las operaciones de offer, add, poll y remove ya explicadas en la interfaz Queue; complejidad lineal para las operaciones de remove(Object) y contains(Object) y tiempo constante para las operaciones de peek, element, y size.

6.2. Deque

Es una interfaz que representa una cola doblemente enlazada, que es una estructura de datos lineal que permite insertar y eliminar elementos por ambos extremos, que engloba las propiedades LIFO y FIFO en una misma estructura. Las clases ArrayDeque y LinkedList implementan a esta interfaz.

6.2.1. ArrayDeque

La clase ArrayDeque implementa dos interfaces, la interfaz Queue y Deque, nos permite implementar la estructura de datos Cola Doblemente Enlazada, pero aplicado sobre un Array, no funciona como las LinkedList por medio de nodos enlazados, pues aquí todos los nodos son contiguos en memoria. Un ArrayDeque tiene las siguientes consideraciones:

- No hay restricciones de capacidad, excepto por la memoria del sistema donde se ejecutan.
- No son sincronizadas, esto significa que no son seguras para ser manipuladas por varios subprocesos
- No se permite el uso de elementos Null

ArrayDeque es más consistente y rápida que un Stack cuando se usa como pila, de igual forma es mejor que LinkedList cuando se utiliza como cola.

7. Mapas

Los mapas en el contexto de la programación no tienen que ver con lo primero que se nos ocurre al escuchar su nombre. El nombre refiere a un concepto más matemático que se refiere a una función que mapea un dominio y un recorrido. Los mapas son conocidos por muchos nombres, entre los que se encuentran diccionario, tabla hash y mapas hash. Todos los nombres anteriores refieren a la misma estructura de datos.

Un mapa mapea o relaciona un conjunto de claves con sus valores asociados de una manera muy parecida a como una función matemática mapea un valor en el dominio al rango. La clave es lo que se debe proporcionar al mapa para buscar un par llave/valor. Las claves en un mapa deben ser únicas, lo que significa que solo puede haber una copia de cada clave en un mapa al mismo tiempo.

Los mapas de cierta manera son parecidos a los conjuntos (set), en el sentido de que el mapa contiene un conjunto de llaves únicas. Como en los conjuntos, se pude buscar una clave, y por lo tanto su valor asociado, en un tiempo cercano a O(1). Esto es debido a que los mapas al igual que los sets utilizan una función hash para almacenar los elementos.

Los mapas, a diferencia de los conjuntos, no almacenan únicamente el valor de clave, sino que el valor asociado también está dentro de la estructura de datos.

Los mapas son especialmente útiles en dos situaciones, cuando se requieren memorizar y cuando se requiere asociar datos. Un ejemplo muy sencillo de primer caso es utilizar un mapa para mejorar la versión recursiva de la sucesión de Fibonacci. La versión original es muy ineficiente ya que requiere calcular múltiples veces la sucesión de Fibonacci de los números. Si se implementa un mapa en el que las claves sean los números y los valores el resultado de calcular la serie de Fibonacci del número, se puede mejorar considerablemente el rendimiento del algoritmo.

Un ejemplo de segundo caso es una relación de ciudades y códigos postales. Si se tiene un programa en el que se debe buscar o almacenar el código postal de diferentes ciudades, se puede crear un mapa en el que las claves sean el nombre de las ciudades y el valor el código postal. De esa manera al ingresar el nombre de una ciudad se podría obtener su código postal en tiempo constante.

7.1. Interfaz Map

Una interfaz es un conjunto de métodos relacionados y vacíos que las clases pueden implementar. Cuando una clase implementa un método, está obligada a implementar los métodos de la interfaz. Las interfaces sirven para formalizar como un objeto interactuará con el exterior y para saber el comportamiento esperado del mismo.

La interfaz Map define objetos que mapean claves y valores. Esta interfaz ofrece tres maneras de verla: como un conjunto de claves, como una colección de valores o como un conjunto de relaciones clave- valor.

El comportamiento de un mapa no está especificado en caso de que se utilice un objeto mutable como clave o que se realice cualquier operación que afecte al resultado del método equals del objeto. Por esta razón, no está permitido que un mapa se contenga como clave a sí mismo.

En general, las clases que implementan la interfaz equals deben tener dos constructores, uno vacío que genera un mapa vacío u otro que recibe un argumento de tipo Map que genera un mapa con las mismas relaciones clave-valor.

Algunos de los métodos más importantes de los mapas son:

- clear, que elimina todas las relaciones del mapa.
- get, que devuelve el valor asociado a una clave.
- put, cuya función es asociar un valor especificado con una llave del mapa.

7.2. Interfaz SortedMap

Esta interfaz provee mapas que están ordenados. El ordenamiento sigue el orden natural de sus claves o a un Comparator suministrado al momento de creación del mapa. Las llaves deben implementar la interfaz comparable y deben ser consistente en el método equals.

Cualquier clase que implemente la interfaz SortedMap debe tener al menos cuatro constructores estándar. Dos de ellos son los que solicita la interfaz Map (que es extendida por esta interfaz) y otros dos son propios de esta interfaz. Uno debe recibir un solo argumento de tipo comparator, que crea un mapa ordenado de acuerdo con el comparador especificado vacío y otro con un argumento de tipo SortedMap, que genera un mapa ordenado con el mismo orden, elementos y comparador del mapa suministrado.

Algunos métodos importantes son:

- firstKey, que devuelve la primera llave del mapa. Hay otro método parecido llamado lastKey que devuelve la última llave.
- subMap, que devuelve un mapa ordenado desde la primera llave suministrada de manera abierta, hasta la segunda suministrada de manera cerrada.
- Values, que devuelve una colección de los valores contenidos en el mapa.

7.3. Interfaz navigableMap

Esta es una interfaz que extiende a SortedMap con métodos de navegación que devuelven al elemento más cercano siguiendo un determinado criterio de búsqueda. Por ejemplo, está el método lowerEntry que devuelve la clave más cercana menor a una clave dada. Los métodos de este mapa deben ser utilizados solo para localizar otros elementos, ya que no están diseñados para permitir modificar las entradas.

Algunos métodos tienen nombre y funcionalidad parecidos a los de la interfaz sortedMap, pero difieren en que permiten elegir si las fronteras o criterios de búsqueda son inclusivos o no. Un ejemplo es el método subMap. Los submapas de un mapa navegable deben implementar la interfaz navigableMap. Los métodos que devuelven entradas no devuelven el valor almacenado, sino que devuelven pares Map.Entry que son una copia de una relación dentro del mapa en el momento de la consulta. Estás entradas deben ser utilizadas para consulta únicamente, ya que las implementaciones no suelen permitir su modificación.

Algunos métodos importantes son:

- pollFirtEntry y pollLastEntry devuelven el mapeo de la primero y última clave respectivamente, y lo eliminan del mapa.
- lowerEntry y higherEntry devuelven el valor más grande estrictamente menor y el valor más pequeño estrictamente mayor a la clave dada respectivamente.
- floorEntry y ceilingEntry hacen lo mismo que los métodos mencionados anteriormente, solo que incluyen a valores iguales a la clave suministrada.

7.3.1. TreeMap

Esta es una implementación de la interfaz NavigableMap que utiliza un árbol rojo-negro. Las claves son ordenadas por su orden natural o por un comparador proporcionado al mapa al momento de creación.

Gracias al árbol rojo-negro, esta estructura puede garantizar tiempo O(log (n)) en las operaciones de búsqueda de una clave, obtención de un valor o clave y de eliminación. Esta implementación no está sincronizada, lo que significa que varios procesos pueden acceder al objeto al mismo tiempo.

Como esta es una implementación de la interfaz NavigableMap, sus métodos son los mismo a los de esa interfaz.

8. Análisis del programa

8.1. Administración

Para comenzar con el análisis del programa, se debe explicar el funcionamiento y propósito de la clase Administración. Esta es la clase fundamental del proyecto, pues contiene al método principal, a las principales colecciones de objetos para su manejo en todas las demás clases y, además, es el puente entre el usuario y el programa en sí.

Es preciso mencionar que, se declararon y utilizaron dos tipos de colecciones como atributos de esta clase, estas fueron List y Map. Se declararon y utilizaron las colecciones de tipo List para poder almacenar ahí a las asignaturas, a los profesores y a los alumnos, cada tipo de dato en su respectiva lista o arreglo dinámico si somos más específicos. El otro tipo de colección que se empleó fue una tabla hash (HasTable), esta se implementó para los grupos existentes en el programa.

La tabla hash almacena los objetos de tipo grupo y a estos les asigna claves al momento de almacenarlos, para poder acceder a ellos por medio de dichas claves. Cualquier objeto que no sea nulo se puede utilizar como clave o como valor. También es importante mencionar que las estructuras descritas anteriormente se definieron como estáticas, esto no solo para poder utilizar a los métodos de las colecciones sin tener que instanciar objetos, sino que también para para facilitar el manejo de estas colecciones en algunas clases ajenas, de esta manera, no es necesario mandar como parámetros a las listas y al mapa por demasiados métodos antes de llegar a su destino.

Una vez dentro del método principal, se instanció un objeto de la clase llamada SubMenus, la cual se detallará más adelante; se definió un ciclo de repetición do-while con una estructura selectiva switch- case para pre-

guntarle constantemente al usuario las acciones que quiera hacer dentro del programa, por lo que, en esencia, la clase Administración funge como un menú. En este menú, se administran a los objetos Alumno, Profesores, Asignatura y Grupo por medio de los métodos de la instancia SubMenus antes mencionada. Ahora es un buen momento para describir a esta clase, la cual cuenta con cuatro atributos importantes: un objeto static de la clase Scanner para ingresar datos; un objeto de la clase Asignar; otro de la clase Crear; y un último de la clase Eliminar. De esta menara se establece la composición de clases, es decir, los objetos de la clase SubMenus tienen como atributos instancias de otras clases para poder trabajar y realizar sus operaciones.

8.2. SubMenus

La clase cuenta con cuatro métodos diferentes, cada uno para controlar lo que le sucede tanto a los alumnos, profesores, grupos como asignaturas. Una de las razones por las cuales esta clase recibe el nombre de SubMenus es porque cada método controla el flujo de entrada, visibilidad, y eliminación de sus objetos correspondientes, es decir, un método es el encargado de administrar a los alumnos, pues por medio de él se pueden ver, crear, eliminar y asociar.

Para implementar este control se utilizaron ciclos de repetición do-while junto con estructuras selectivas switch-case para escoger el caso correspondiente.

Una vez dentro del caso elegido, se utilizan los métodos de los objetos previamente instanciados para poder acceder a sus funcionalidades; esto aplica para cualquier método en cualquier caso escogido de la estructura switch-else. Por ejemplo, para poder crear un alumno nuevo, nos vamos al submenú de alumnos y seleccionados la opción Crear alumno, dentro del programa nos encontraríamos en alguno de los casos de la estructura selectiva switch, y desde aquí se utilizaría al método CrearAlumno del objeto de la clase Crear para poder instanciar un objeto de la clase Alumno. Todo esto se realiza en el programa pasando como parámetros las listas o las colecciones correspondientes. En esencia, este es el accionar de la clase SubMenus, poder brindar un control más fluido y entendible al usuario.

8.3. Crear

La clase Crear se diseñó para que sus métodos fueran llamados por la clase SubMenus y fueran capaces de crear e inicializar a los diferentes objetos

de los distintos tipos de dato definidos y, de esta manera, ingresar estos objetos en sus respectivas colecciones. Es preciso mencionar que esta clase cuenta con una gran cantidad de interacciones con objetos de otras clases, pues maneja a objetos de las clases Alumno, Profesor, Asignatura, Grupo y Dirección de forma directa, ya que es la que crea e inicializa estos objetos.

En los métodos de esta clase, se le piden al usuario los datos de los objetos pertinentes, por ejemplo, si estamos creando a un nuevo profesor, se piden sus datos como nombre, apellido, número de cuenta, entre otros; e incluso en los métodos se diseñaron algunas restricciones para no ingresar alumnos con el mismo número de cuenta, o no ingresar grupos con la misma clave. Esto brinda una mayor organización y encapsulación en el programa.

8.4. Asignar

Otra de las clases definidas en este programa fue la que tiene por nombre Asignar. En esta clase, se tienen dos métodos que sirven para asociar tanto a alumnos como a profesores con sus respectivos grupos.

Se pensó en crear esta clase especialmente para la asociación de alumnos y profesores, puesto que nuestro pensamiento abstracto percibió a esta funcionalidad como un procedimiento fundamental en común para ambos tipos de dato. De esta manera, el programa queda organizado por bloques que realizan la misma funcionalidad para diferentes objetos. La asociación que se realiza a un alumno como a un profesor se realiza de la siguiente manera: los métodos reciben el mapa de grupos para poder ingresar ahí al alumno o al profesor, dependiendo el caso, pero antes se verifica que ya se hayan creado grupos, en caso de que no, se retorna al submenú correspondiente; en caso contrario, se muestran en pantalla los diferentes grupos existentes y se le pide al usuario que ingrese la clave del grupo al que le quiera asignar el profesor o el alumno.

Si todo lo anterior se lleva a cabo, se crean "copias" del grupo seleccionado, almacenado en el mapa, y se guardan en las colecciones propias de cada alumno y de cada profesor. De la misma manera, se crean copias de los alumnos y los profesores para guardarlos en el objeto grupo y en el objeto asignatura, para que todos los objetos sepan que objetos tienen.

8.5. Imprimir

Esta clase fue de suma importancia para mostrar el contenido de las modificaciones realizadas al programa, y para mostrar el ingreso de datos, así como el estado de los atributos de los objetos. Por lo tanto, se decidió definir

a los métodos como estáticos, pues no consideramos necesario instanciar objetos de esta clase para poder imprimir en pantalla los estados de los objetos. Así, la impresión se logra más fácilmente en diferentes lugares del programa.

Los métodos definidos se encargaron de mostrar los atributos de los cuatro diferentes tipos de dato en dos versiones: la primera versión para cada tipo de dato imprime las características recortadas o básicas del objeto; mientras que la segunda versión imprime los datos con mayor detalle.

Para lograr esto, se utilizaron instancias de las clases Alumno, Profesor, Grupo y Asignatura para usar sus métodos de acceso, y así lograr mostrar sus atributos; también se utilizaron a las colecciones correspondientes, las cuales ingresaron como parámetro a los métodos, para poder iterar sobre ellos en el caso de que se requieran visualizar todos los elementos de dichas colecciones. Esto resulta útil en los diferentes menús, pues es necesario imprimir en pantalla la información para que el usuario sea capaz de decidir qué hacer.

8.6. Eliminar

La clase eliminar contiene métodos que sólo pueden ser llamados por la clase SubMenú cuando nuestro usuario desea eliminar algún Grupo, Alumno, Profesor o Asignatura en su totalidad. Todos los métodos reciben como parámetro la List o el Hash Set (sólo para los grupos) principal que se encuentra en la clase Administración, estos parámetros son colecciones de objetos del mismo tipo del que se va a eliminar. Tenemos cuatro métodos en total, y todos incluyen una verificación para saber si la List o el Hash Set recibido son vacíos, si esto es cierto se muestra al usuario un mensaje y termina la ejecución de estos métodos.

Pero si al menos tienen algún elemento, se manda llamar un método específico para imprimir el contenido de la colección, esto para que el usuario pueda visualizar mejor su elección a la hora de eliminar un objeto; se le pregunta al usuario el número de profesor, alumno o asignatura, o la clave del grupo a eliminar, posteriormente se solicita el método eliminar específico para el tipo de objeto elegido desde su respectiva clase, en cualquiera de los métodos nos apoyamos con un objeto temporal llamado “eliminar” para solicitar el método de eliminación de las clases.

8.7. Grupo

En el diseño de nuestro proyecto, decidimos que la clase Grupo sería una de las más importantes puesto que tiene varios objetos como atributos, una

instancia de esta clase puede conocer a los alumnos que se inscribieron en ese grupo, al profesor encargado y la asignatura que se imparte. Adicionalmente incluye una clave, que es su identificador único como grupo, aprovechando las propiedades de una Hash Table, en nuestra clase principal Administración, decidimos implementar esta estructura de datos para instancias de Grupo, ya que podemos acceder a ellos por medio de su clave.

Esta clase emplea un constructor no vacío que inicializa los atributos de encargado (Profesor), clase y asignatura; entonces, para poder instanciar un Grupo tenemos que crear un Profesor y una Asignatura forzosamente antes de, de lo contrario tendremos algunos errores que en la Clase Crear tenemos verificados, para que sólo muestren un mensaje al usuario, sin detener la ejecución. Para poder manipular la el ArrayList de alumnos inscritos a un Grupo, tenemos un método que recibe un alumno y lo añade por medio del método *.add, de igual manera tenemos un método para eliminar alumnos inscritos con el método *.remove. Para manipular tanto el profesor como la asignatura, tenemos dos métodos para cada uno: uno hace nulo al atributo y el otro permite cambiarle el valor. Al igual que las demás clases, tenemos un método de eliminación, esta vez retira un grupo del Hash Table principal, por medio de su clave. Y dos métodos de impresión, uno resumido (clave, asignatura y profesor) y otro completo, que muestra todos los atributos de un Grupo.

8.8. Asignatura

La clase Asignatura permite instanciar una asignatura, tiene como atributos nombre, créditos, clave y también conoce al conjunto de grupos que la pueden impartir. Contiene un constructor no vacío que inicializa todos sus atributos exceptuando los grupos. Dado que una Asignatura conoce los grupos que la imparten, tiene métodos para eliminar o agregar grupos a su conjunto. Incluye un método de búsqueda que permite verificar si en la lista principal de Asignaturas de la clase Administración, ya existe una asignatura que tenga determinada clave, este método es útil para verificar que todas las asignaturas que sean creadas tengan una clave única. También incluimos un método para eliminar una Asignatura en la lista principal por medio de su clave. Para poder visualizar los atributos de una asignatura, tenemos un método que imprime de forma resumida: nombre, créditos y clave; y otro método que imprime todos sus atributos incluyendo grupos.

8.9. Alumno

La clase Alumno representa a un Alumno con atributos de identidad (nombre completo, edad, número de cuenta), contiene una Dirección, así como un Hash Set de los Grupos a los que esté inscrito (no pueden ser más de tres) y una variable estática de control para saber cuántos alumnos hemos creado a lo largo del programa. Incluye un constructor no vacío que puede inicializar todos sus atributos exceptuando a los Grupos. Dado que un Alumno puede conocer los grupos a los que está inscrito, se incluyen métodos para ingresar o retirar grupos de su Hash Set con las claves específicas.

Se tiene un método de búsqueda que permite realizar una verificación cuando es utilizado en la clase Crear, pues siempre de que creamos un nuevo Alumno, buscamos en la Lista principal de Alumnos de la clase Administración por algún alumno con el mismo número de cuenta, si ya existe alguno, entonces no se puede crear ese nuevo Alumno. También tenemos un método que permite eliminar Alumnos en la Lista principal de Alumnos, así como de los grupos a los cuales esté inscrito. Cuando se necesiten visualizar los atributos de un alumno, tenemos dos variantes de un método de impresión: para visualizar sólo su nombre, apellido y número de cuenta. O para visualizar todos sus atributos, incluyendo su dirección y grupos, para estos últimos se emplean los métodos de impresión presentes en las clases de Dirección y Grupo.

8.10. Profesor

La clase Profesor representa un profesor con atributos de identidad: nombre completo, edad y número de cuenta, así como un objeto de tipo Dirección y un Hash Set (conjunto) con los grupos en donde imparte clases, se optó por un conjunto ya que un grupo puede ser identificado por medio de una clave y además no se pueden repetir. Cuenta con un constructor vacío que inicializa todos sus atributos excepto el Hash Set de grupos, para estos últimos tenemos dos métodos que añaden o retiran Grupos del conjunto, por medio de los métodos *.add y *.remove respectivamente. La clase contiene un método para realizar búsqueda en la lista principal de profesores, esto por si al momento de crear un profesor con determinado número de cuenta, y el número de cuenta ya existe, no se permite al usuario duplicar esa información. Al igual que con la clase alumno, tenemos métodos que imprimen solamente losa tributos como nombre y número de cuenta, o los atributos completos de un Profesor. Incluye también un método de eliminación, para retirarlo tanto de la lista principal de Profesores como de los grupos donde

impartía clase.

8.11. Dirección

La clase Dirección contiene los atributos necesarios para instanciar el domicilio de un profesor o alumno. Cuenta con los atributos de estado, municipio, colonia, número y calle. Consta únicamente de un constructor no vacío que inicia todos los atributos de un objeto de esta clase, y un método para poder imprimirlos.

9. Conclusiones individuales

9.1. Argüello, Dante

La cantidad de elementos teóricos y prácticos involucrados en el desarrollo de este proyecto complementan una experiencia bastante satisfactoria para el aprendizaje que logré con este Proyecto. Fué agradable trabajar con mis compañeros de manera virtual, pues nos apoyamos de herramientas que facilitaron nuestra causa, como crear un repositorio en GitHub, las reuniones por videollamada y los sistemas de almacenamiento en la nube, no dudo que estas herramientas son útiles ya en campo laboral cuando se debe desarrollar un proyecto con developers de distintas partes del mundo.

Durante el desarrollo nos surgieron bastantes retos lógicos, y en mi caso algunos retos teóricos, y entre todos estuvimos dispuestos a ayudarnos con lo que no entendíamos. Siento que el desarrollo de nuestro Proyecto fue a la par del avance en teoría vista en clase, se logró la implementación correcta de las Colecciones gracias al aporte de todos, pues al ser estructuras de datos nuevas (o al menos en mi caso), la investigación de cada uno de los integrantes fue puliendo la forma en cómo implementábamos los conjuntos principalmente, las listas y los mapas.

De manera personal, siento que no tuve grandes aportaciones prácticas durante el desarrollo del proyecto a diferencia de mis compañeros, pero todos estuvimos dispuestos a trabajar con alguna parte del proyecto y tratamos de equilibrar las cargas de trabajo. Siento que cumplí con el objetivo ya que aprendí a implementar algunas de las colecciones más importantes, y me agradó nuestro sistema de trabajo. Adicionalmente se fomentó la investigación, y la búsqueda de herramientas prácticas.

9.2. López, Ricardo

Este proyecto fue muy importante para la comprensión de colecciones y sobre todo, para el trabajo en equipo. La parte en donde más aprendí sobre las colecciones fue cuando estaba investigando sobre ellas y donde colocarlas. Aunque en este programa decidimos usar algunas de las versiones más simples de cada tipo de colección, durante la investigación se pudo apreciar que hay una gran variedad de ellas.

Muchas de las colecciones manejan conceptos que eran totalmente desconocidos para nosotros, como algunos objetos y estructuras de datos especiales que utilizan algunas colecciones o la sincronización.

Después de la investigación, ya en el momento de implementar las estructuras en el programa, se presentaron algunos retos. Por ejemplo, iterar en un mapa no es tan sencillo como en otras estructuras, y, aunque tampoco es muy complicado, no llevo a investigar cómo hacerlo.

El uso de conjuntos fue totalmente nuevo en este proyecto también. Por la simplicidad de los conjuntos, fue difícil encontrar un lugar para ellos; pero si no hubiera sido por el proyecto, no sabríamos de su existencia y de lo relacionados que están con los mapas.

Con respecto al trabajo en equipo, fue una experiencia diferente a lo que esperaba. Creí que se iba a complicar mucha la comunicación entre los integrantes y que iba a llevar a muchos mal entendidos. La realidad fue que nos pudimos comunicar bien, repartir el trabajo entre todos y sobre todo complementarnos los unos a los otros.

9.3. Sánchez, Marco

Después de haber terminado el proyecto uno de la asignatura de programación orientada a objetos, puedo concluir que se logró cumplir el objetivo trazado al comienzo de manera satisfactoria. Esto es debido a que, tanto el equipo en conjunto como cada integrante en forma particular, conseguimos conocer, comprender e implementar, en un programa funcional, los principales aspectos teóricos y prácticos de las colecciones; así como sus variantes y aplicaciones en el lenguaje de programación Java. Esto resultó muy importante, ya que las colecciones son poderosas herramientas para trabajar con datos, objetos y estructuras dentro de los programas, y así poder resolver los problemas propuestos en ellos. Así mismo, se lograron aplicar y poner en práctica los conceptos básicos revisados en clase, que tienen que ver con la programación orientada a objetos, para apropiarnos satisfactoriamente de los conocimientos adquiridos en esta primera parte del curso. Por último,

pero sin dejar de lado su importancia, debo mencionar que se fomentó y desarrolló el trabajo en equipo a distancia, pues el proyecto se llevó a cabo con una gran coordinación y cooperación por parte de los integrantes. Los principales conflictos y las diferentes opiniones de cada integrante se discutieron y se solucionaron, lo cual benefició el avance del proyecto y la apropiación de conocimiento significativo. Esta actividad me pareció un poco sofocante por la carga de trabajo, pero fue muy enriquecedora en cuanto a los contenidos y al aprendizaje obtenido.

10. Conclusión General

Este proyecto nos ha sido de suma importancia para poner en práctica y seguir repasando los conceptos vistos en la materia de POO en cuanto a Colecciones, Utilerías, Paquetes y uso del lenguaje Java en general, en conjunto con teoría vista en EDA 2. Creemos que el objetivo de este proyecto se ha cumplido ya que pudimos implementar varias de las colecciones más importantes presentes en Java, así como sus métodos y utilerías; como equipo tuvimos un correcto desarrollo desde el primer día, con comunicación constante ya sea por mensajería instantánea o por reuniones en Google Meet. Dado que uno de los objetivos era fomentar el trabajo en equipo a distancia, y en un contexto donde la virtualización ha tenido auge aunado a un proyecto en el campo de la programación, decidimos aprender a utilizar la herramienta de control de versiones Git (por medio de GitHub), situación que nos facilitó bastante el trabajo en equipo.

Entre las experiencias más importantes que nos llevamos de este Proyecto es que a partir de los conceptos de Programación Orientada a Objetos, así como del nivel de abstracción de cada uno de los integrantes, pueden existir infinitas formas de diseñar un programa, por lo que llegar a un consenso a la hora de definir clases, objetos, composiciones y atributos es un esfuerzo en el que todos debemos contribuir, donde es importante una eficaz comunicación entre los integrantes, para lograr un trabajo más consistente y donde todos tengan un rol en el desarrollo. En cuanto al desarrollo práctico del Proyecto, nuestra experiencia más grata fue la implementación del control de versiones, así como observar que en cada clase de teoría aprendíamos más cosas que implementábamos para hacer más robusto nuestro trabajo, y la implementación de un editor de texto nuevo (Latex). De igual manera, el reto de establecer algunas de las Colecciones como HashTable y HashMap nos hizo comprender cabalmente su funcionamiento para poder aportar ideas de implementación. No visualizamos alguna desventaja en los contenidos de

este proyecto, exceptuando que en situaciones específicas se puede volver complejo un diseño de programa orientado a objetos. Se considera que la calidad de este proyecto fue correcta para implementar lo visto en clase y laboratorio y mantener un aprendizaje continuo.

Referencias

[1] Oracle. Java Platform Standard Edition & Java Development Kit Version 11 API Specification, 2018

[2] Lee, Kenn & Hubbard, Steve. Data Structures and Algorithms with Python, 2015