MÉTODO DE LA INGENIERÍA:

**Contexto problemático:**

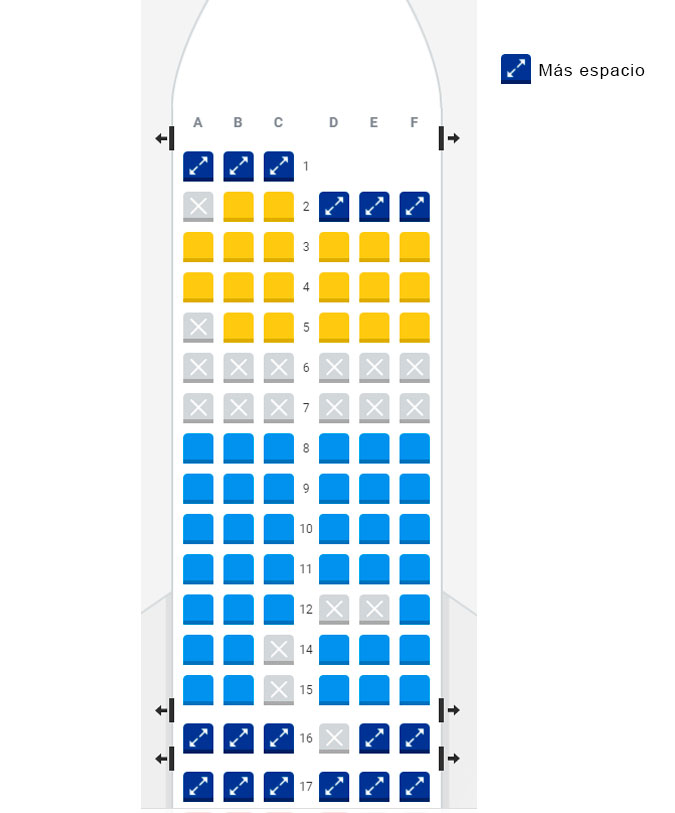
Después del excelente rendimiento en sus trabajos académicos, usted y sus compañeros han sido seleccionados por una reconocida aerolínea para realizar una primera versión de un sistema cuyo objetivo principal es mejorar el orden en el proceso de ingreso y salida del avión.

Para la versión finalizada del sistema se cuenta con una base de datos. Sin embargo, para esta primera versión se espera simplemente que dicha base de datos sea simulada mediante un cargue inicial de pasajeros al sistema por medio de algún archivo de texto plano (Debe ser generado por ustedes también). Es decir, dado un vuelo de la aerolínea, el sistema debe permitir cargar la información de los pasajeros correspondiente a ese vuelo.

Ahora bien, una vez un pasajero llegue a la sala de abordaje que le corresponde para esperar su vuelo, se debe buscar, de la manera más eficiente posible (pues a futuro la cantidad de datos será significativamente muy grande) su información completa y registrar su llegada a la sala. Este último paso es muy importante, pues se espera premiar la puntualidad de los pasajeros ingresando al avión en el orden de llegada (aunque teniendo en cuenta que se sigue manteniendo el llamado por secciones del avión, empezando por las más alejadas a la puerta de ingreso hasta la más cercana a ella). Con lo cual, una vez se abran las puertas del avión, el sistema debe permitir mostrar, al miembro de la tripulación encargado, en qué orden deben ingresar los pasajeros.

Sin embargo, y en pro de seguir buscando mejorar la relación con sus mejores clientes, la aerolínea quiere que la primera clase tenga algunas reglas especiales cuando sean llamados para el ingreso al avión, es decir, no solo teniendo en cuenta su orden de llegada, sino priorizando también otros datos como millas acumuladas, atención especial requerida, tercera edad u otros datos relevantes que a su equipo se les pueda ocurrir.

Además de lo anterior, la salida del avión también debe realizarse en un orden establecido teniendo en cuenta la siguiente configuración que poseen sus aviones:



En donde, quienes primero salen son aquellos que se encuentran en las primeras filas y para cada fila el orden está establecido por cercanía al pasillo u orden de llegada como última instancia. De nuevo, la persona de la tripulación de vuelo a cargo del sistema podrá ver en qué orden deben salir los pasajeros.

**Paso 1: Identificación del problema:**

* Implementar un sistema que permita cargar la información de los pasajeros de forma eficiente.
* Establecer un orden para el ingreso y salida del avión que permita mejorar el proceso y evitar problemas de organización.
* Priorizar el ingreso de los pasajeros de primera clase teniendo en cuenta no solo el orden de llegada sino otros datos relevantes como millas acumuladas, atención especial requerida, tercera edad u otros.
* Garantizar la precisión en la ordenación de los pasajeros y su registro de llegada a la sala correspondiente.
* Necesidad de una solución que permita cargar la información de los pasajeros de forma eficiente y establecer un orden adecuado para mejorar la experiencia de los pasajeros.

**Paso 2: Recopilación de información:**

* Aerolínea: Empresa que se dedica al transporte de pasajeros y/o carga a través del uso de aeronaves.
* Base de datos: Conjunto de datos organizados y estructurados que se almacenan en un sistema informático para su posterior consulta y uso.
* Sala de abordaje: Área dentro del aeropuerto donde los pasajeros esperan para abordar el avión.
* Cargue inicial: Proceso de carga inicial de datos en el sistema, en este caso de los pasajeros que abordaron el avión.
* Eficiencia: Capacidad de realizar una tarea de manera óptima y con el menor esfuerzo posible.
* Registro de llegada: Proceso de ingresar en el sistema la información de los pasajeros una vez que llegan a la sala de abordaje.
* Puntualidad: Característica de ser exacto o cumplir con el tiempo establecido.
* Llamado por secciones: Proceso de llamado de los pasajeros por grupos preestablecidos, generalmente por secciones del avión.
* Orden de llegada: Secuencia de llegada de los pasajeros a la sala de abordaje.
* Primera clase: Clase de servicio superior en la que los pasajeros reciben mayor comodidad y atención que en otras clases.
* Millas acumuladas: Puntos que los pasajeros acumulan al viajar en la aerolínea, los cuales pueden ser canjeados por beneficios adicionales.
* Atención especial: Servicios adicionales que la aerolínea puede ofrecer a los pasajeros que lo necesiten, como asistencia para discapacitados o menores no acompañados.
* Tercera edad: Personas mayores de 60 años.
* Salida del avión: Proceso de salida ordenada de los pasajeros del avión.
* Configuración: Disposición y distribución de los asientos dentro del avión.
* Fila: Serie de asientos colocados en forma horizontal dentro del avión.
* Orden de salida: Secuencia en la que los pasajeros deben salir del avión, establecida por la configuración de los asientos.

**Paso 3: Búsqueda soluciones creativas:**

Alternativas Lectura:

1. Usar la clase Scanner: La clase Scanner se utiliza para leer datos en Java. Esta clase proporciona una gran cantidad de métodos para leer diferentes tipos de datos, como números, cadenas de texto, etc. Para utilizar Scanner, se debe crear una instancia de la clase Scanner y pasarle como argumento un objeto de la clase File que representa el archivo de texto a leer. Luego, se puede utilizar los métodos nextLine() o next() para leer una línea o una palabra a la vez.
2. Usar FileReader y BufferedReader: La clase BufferedReader se utiliza para leer datos de un archivo de texto de manera eficiente. Esta clase utiliza un búfer para leer una cantidad específica de datos desde el archivo y reducir el número de accesos al disco. Es muy útil para leer grandes cantidades de datos de manera eficiente. Para utilizar BufferedReader, se debe crear una instancia de la clase FileReader y pasarla como argumento al constructor de BufferedReader. Luego, se puede utilizar el método readLine() para leer una línea de texto a la vez.

Alternativas deserialización:

1. Libreria Gson: La librería Gson proporciona una clase llamada Gson que se utiliza para realizar la deserialización. Esta clase ofrece varios métodos que permiten la deserialización de objetos JSON en objetos Java. El método más utilizado para la deserialización es el método fromJson(), que convierte una cadena JSON en un objeto Java de la clase especificada.
2. Jackson: Una librería de procesamiento de JSON de alto rendimiento y ampliamente utilizada. Es conocida por su capacidad para manejar JSON grande y complejo de manera eficiente.
3. org.json: Una librería simple y liviana para el procesamiento de JSON. Es fácil de usar y está diseñada para trabajar con JSON simple y estructurado.

Alternativas Búsqueda Pasajeros:

1. TreeMap: Se puede utilizar un TreeMap si se necesita almacenar y recuperar elementos asociados a una clave de forma ordenada. En un TreeMap, los elementos se ordenan automáticamente por clave, lo que puede ser útil si se necesita acceder a los elementos en un orden específico.
2. ArrayList: Se puede utilizar un ArrayList si se necesita almacenar y acceder a elementos de forma secuencial. En un ArrayList, los elementos se almacenan en un orden específico y se puede acceder a ellos por índice.
3. HashSet: Se puede utilizar un HashSet si se necesita almacenar y recuperar elementos de forma eficiente sin repetición. En un HashSet, los elementos se almacenan sin ningún orden específico, pero se puede buscar un elemento dado de forma muy eficiente.
4. Hash Table: Se puede utilizar una tabla hash si se necesita almacenar y recuperar elementos asociados a una clave de forma eficiente. En una tabla hash, los elementos se almacenan en una estructura de tabla que permite acceder a ellos rápidamente mediante su clave. Esta estructura es especialmente útil cuando se necesita buscar elementos de forma muy eficiente.

Alternativa registro de llegada:

1. Queue (Cola): Una cola es una estructura de datos que funciona en el principio de "primero en entrar, primero en salir". Se puede utilizar una cola para mantener un registro de los pasajeros que han llegado a la estación de aviones en el orden en que llegaron. Luego, al abordar el avión, se puede utilizar la cola para determinar el orden de abordaje, teniendo en cuenta las secciones del avión.
2. Priority Queue (Cola de prioridad): Una cola de prioridad es una variante de la cola donde los elementos tienen una prioridad asociada. Se puede utilizar una cola de prioridad para mantener un registro de los pasajeros que han llegado a la estación de aviones en función de su prioridad, como su puntualidad o su categoría de viajero frecuente. Luego, al abordar el avión, se puede utilizar la cola de prioridad para determinar el orden de abordaje.
3. LinkedList (Lista enlazada): Una lista enlazada es una estructura de datos que se utiliza para almacenar una secuencia de elementos en orden. Se puede utilizar una lista enlazada para mantener un registro de los pasajeros que han llegado a la estación de aviones en el orden en que llegaron. Luego, al abordar el avión, se puede utilizar la lista enlazada para determinar el orden de abordaje, teniendo en cuenta las secciones del avión.
4. TreeMap: Un TreeMap también puede utilizarse para mantener un registro de los pasajeros en función de su hora de llegada, por ejemplo, utilizando la hora como clave y el registro del pasajero como valor. Al abordar el avión, se pueden recuperar los registros de los pasajeros en orden de llegada utilizando las claves del TreeMap.
5. ArrayList: Un ArrayList puede utilizarse para mantener un registro de los pasajeros en el orden en que llegaron. Luego, al abordar el avión, se puede utilizar el ArrayList para determinar el orden de abordaje, teniendo en cuenta las secciones del avión. Sin embargo, es importante tener en cuenta que eliminar elementos de un ArrayList puede ser ineficiente, lo que podría ser un problema si los pasajeros se presentan en la estación de aviones en grandes cantidades.

Alternativa entrada y salida:

1. PriorityQueue (Cola de prioridad): Una cola de prioridad es una estructura de datos que ordena los elementos según una prioridad determinada. En este caso, se podría utilizar una cola de prioridad para ordenar los pasajeros según la hora de llegada, categoría de pasajero, millas acumuladas, atención especial requerida, tercera edad, etc. de manera que se garantice que los pasajeros con mayor prioridad (por ejemplo, los de primera clase, o los que requieren atención especial) tengan un orden de abordaje preferencial, pero siempre respetando las secciones del avión.
2. TreeMap: Un TreeMap es una estructura de datos que mantiene un mapa ordenado según las claves. En este caso, se podría utilizar un TreeMap para almacenar información de los pasajeros, por ejemplo, utilizando la hora de llegada como clave y un objeto que contenga los datos relevantes del pasajero como valor. De esta manera, se puede mantener un registro ordenado de los pasajeros en función de la hora de llegada, pero teniendo en cuenta los otros factores, como la categoría de pasajero, millas acumuladas, atención especial requerida, tercera edad, etc.
3. ArrayList: Un ArrayList es una estructura de datos que permite el acceso secuencial a sus elementos y se puede utilizar para almacenar los pasajeros en orden de llegada, pero luego se puede ordenar utilizando la hora de llegada, categoría de pasajero, millas acumuladas, atención especial requerida, tercera edad, etc.
4. HashMap: Un HashMap es una estructura de datos que mantiene un mapa de pares clave-valor. En este caso, se podría utilizar un HashMap para almacenar información de los pasajeros, por ejemplo, utilizando su número de identificación como clave y un objeto que contenga los datos relevantes del pasajero como valor. De esta manera, se puede acceder rápidamente a la información de un pasajero específico y ordenarlos según diferentes factores.
5. LinkedList: Una LinkedList es una estructura de datos que permite el acceso secuencial a sus elementos y también se puede utilizar para almacenar los pasajeros en orden de llegada, pero luego se puede ordenar utilizando la hora de llegada, categoría de pasajero, millas acumuladas, atención especial requerida, tercera edad, etc. Además, se pueden eliminar elementos de una LinkedList de manera eficiente, lo que podría ser útil si se presentan cambios en el orden de abordaje.

**Paso 4: Transición de las Ideas a los Diseños Preliminares**

En la lectura de datos no descartamos ninguna inicialmente, será necesario de un análisis más profundo para escoger la mejor opción.

La revisión cuidadosa de las alternativas nos conduce a los siguiente:

BufferedReader y FileReader:

- Ofrecen una forma eficiente de leer datos de un archivo de texto, reduciendo el número de accesos al disco y mejorando el rendimiento del programa.

- Son parte del lenguaje Java, por lo que no es necesario agregar librerías externas al proyecto.

- Son fáciles de aprender y utilizar, lo que los hace ideales para usuarios nuevos en el desarrollo de software..

- No proporcionan tantas opciones de personalización como algunas alternativas externas.

- Pueden requerir más código para implementar que otras alternativas, especialmente si se desea leer datos de forma estructurada.

Scanner:

* El uso de un scanner de Java para la lectura de datos de un archivo de texto plano es muy fácil y conveniente. Solo se necesita crear una instancia del scanner y usar sus métodos para leer los datos.
* El scanner de Java es muy flexible, ya que puede leer diferentes tipos de datos, como enteros, flotantes, cadenas y caracteres, lo que lo hace útil para leer diferentes tipos de archivos de texto plano.
* El scanner de Java maneja automáticamente muchos errores que podrían ocurrir durante la lectura de datos, como errores de formato o de entrada/salida, lo que simplifica el proceso de lectura de archivos.
* El scanner de Java puede ser menos eficiente en términos de rendimiento en comparación con otras formas de lectura de archivos, como la lectura de archivos por lotes. Esto se debe a que el scanner realiza operaciones adicionales para determinar el tipo de dato que se está leyendo.
* El scanner de Java puede no ser la mejor opción para leer archivos de texto plano muy grandes, ya que puede ser lento y requerir grandes cantidades de memoria.
* El scanner de Java puede no ser la mejor opción para leer datos estructurados en archivos de texto plano, como datos separados por comas o tabulaciones, ya que su capacidad para manejar estos formatos es limitada y puede requerir más trabajo para procesar los datos.

En la deserialización de datos, descartamos las opción de Jackson, debido a la complejidad de su implementación respecto a las otras 2 opciones.

La revisión cuidadosa de las otras alternativas nos conduce a los siguiente:

Librería Gson:

* Facilita la conversión de datos en formato JSON a objetos Java y viceversa.
* Permite la personalización de la conversión mediante la creación de adaptadores personalizados.
* Tiene una documentación clara y completa, lo que facilita su uso y aprendizaje.
* Es una librería ampliamente utilizada y respaldada por la comunidad.
* Puede ser menos eficiente que otras alternativas para la conversión de datos debido a la sobrecarga de reflexión y el uso de memoria.
* Puede ser más complejo de utilizar en comparación con otras alternativas, especialmente para usuarios nuevos en el desarrollo de software.

JSONObject:

* Es una librería simple y liviana para el procesamiento de JSON en Java.
* Proporciona una API fácil de usar para trabajar con JSON simple y estructurado.
* Es compatible con varias estructuras de datos Java, incluyendo List, Map y Array.
* Es una de las librerías más populares para trabajar con JSON en Java.
* No es adecuada para manejar JSON complejo o grande debido a su simplicidad.
* La documentación y la comunidad de soporte son relativamente limitadas en comparación con otras librerías de procesamiento de JSON en Java.
* Puede ser menos eficiente en términos de tiempo de ejecución en comparación con otras librerías de procesamiento de JSON en Java.

En la búsqueda de pasajeros, descartamos TreeMap, pues su orden por claves es innecesario para resolver este problema, además, al tener una estructura de árbol lo hace ineficiente al usar datos muy grandes.

descartamos arraylist, pues su búsqueda está condicionada a la cantidad de datos, para mantener una búsqueda eficiente, es necesario mantener una lista de datos pequeña.

La revisión cuidadosa de las otras alternativas nos conduce a los siguiente:

HashSet:

* Proporciona una alta eficiencia para operaciones como la inserción, eliminación y búsqueda de elementos.
* Almacena los elementos de forma no ordenada, lo que puede ser beneficioso en algunos casos.
* Es fácil de usar y entender.
* No permite el acceso a los elementos por su índice, lo que puede ser un problema en algunos casos.
* No almacena los elementos ordenados, lo que puede ser un problema en algunos casos.
* No permite elementos duplicados, lo que puede ser un problema en algunos casos.

Hash table:

* Proporciona una alta eficiencia para operaciones como la inserción, eliminación y búsqueda de elementos.
* Almacena los elementos de forma no ordenada, lo que puede ser beneficioso en algunos casos.
* Proporciona una API rica para manipular los elementos almacenados.
* No permite el acceso a los elementos por su índice, lo que puede ser un problema en algunos casos.
* No almacena los elementos ordenados, lo que puede ser un problema en algunos casos.
* No permite elementos duplicados, lo que puede ser un problema en algunos casos.

En la entrada y salida de pasajeros, descartamos HashMap, Arraylist y Linkedlist, porque no cuentan con una condición de orden integrada, lo cual implicaría escribir más código para organizar a los pasajeros.

La revisión cuidadosa de las otras alternativas nos conduce a los siguiente:

PriorityQueue:

* Permite almacenar elementos en un orden específico, definido por la prioridad de cada elemento.
* Las operaciones de inserción y extracción son muy eficientes y tienen una complejidad de tiempo de O(log n).
* Se puede implementar como un heap binario, lo que permite una implementación eficiente.
* No permite el acceso directo a un elemento específico. La única forma de acceder a los elementos es a través de la cola.

TreeMap:

* Al igual que PriorityQueue, permite almacenar elementos en un orden específico, definido por una clave en lugar de una prioridad.
* Permite acceso directo a los elementos a través de la clave.
* Implementado como un árbol de búsqueda binaria, lo que permite una implementación eficiente.
* Soporta operaciones de búsqueda, inserción y eliminación en O(log n) tiempo.
* No permite duplicados. Si se intenta insertar un elemento con una clave que ya está en el mapa, no se agregará.
* Las operaciones de inserción y eliminación son ligeramente más lentas que en HashSet o ArrayList, aunque todavía son muy eficientes.

**Paso 5: Evaluación y Selección de la Mejor Solución**

Criterios

Lectura de datos:

Criterio A: Eficiente al leer datos estructurados. Se prefiere una solución con mejor eficiencia que las otras consideradas. La eficiencia puede ser:

[4] Constante

­[3] Mayor a constante

­[2] Logarítmica

­[1] Lineal

Criterio B: Familiaridad con la solución. Se prefiere una solución con la que el equipo sea familiar. La solución puede ser:

[3] Familiar

­[2] Conocida

­[1] No conocida

Criterio C: Función independiente. Se prefiere una solución que no requiera de mas librerias

[3]. No requiere de librerías adicionales para funcionar.

[2]. Requiere solo de 1 librería adicional para funcionar

[1]. Requiere de más de 1 librería adicional para funcionar

Evaluación

Evaluando los criterios anteriores en las alternativas que se mantienen, obtenemos la siguiente tabla:

|  | Criterio A | Criterio B | Criterio C | Total |
| --- | --- | --- | --- | --- |
| Scanner | Lineal  1 | Familiar  3 | 1 librería adicional  2 | 6 |
| Bufferedreader | Lineal  1 | Conocida  2 | Más de 1 librería  1 | 4 |

­

Deserialización de datos:

Criterio A: Completitud. Se prefiere una solución que logre el objetivo de convertir un texto Json a objetos de java.

­[3] Puede convertir cualquier archivo de Json en un objeto de Java

­[2] Puede convertir algunos archivos de Json en algunos objetos de Java

[1] No puede convertir ningún archivo Json en objetos de Java

Criterio B:Facilidad de usar. Se prefiere una solución que sea fácil de usar

[2] Se puede deserializar datos con solo 1 linea de comando

[1] Se puede deserializar datos con más de 1 línea de comando

Criterio C: Eficiencia para datos pequeños. Se prefiere una solución que tenga mayor eficiencia con datos pequeños en el cargue inicial.

[4] Constante

­[3] Mayor a constante

­[2] Logarítmica

­[1] Lineal

|  | Criterio A | Criterio B | Criterio C | Total |
| --- | --- | --- | --- | --- |
| Gson | Cualquier archivo  3 | 1 Línea  2 | Lineal  1 | 6 |
| JSONObject | Algunos archivos  2 | 1 Línea  2 | Lineal  1 | 5 |

Búsqueda de pasajeros

Criterio A: Eficiencia. Se prefiere una solución eficiente. La eficiencia puede ser:

[4] Constante

­[3] Mayor a constante

­[2] Logarítmica

­[1] Lineal

Criterio B: Sostenibilidad. Se prefiere una solución que sostenga su rendimiento mientras más datos tiene:

­­[5] No pierde eficiencia sin importar los datos que tenga

[4] Casi no pierde eficiencia con una cantidad grande de datos

­[3] Tiene una pérdida moderada de eficiencia con una cantidad grande de datos

[2] Tiene una pérdida significativa de eficiencia con una cantidad grande de datos

[1] Tiene una pérdida total de eficiencia con una cantidad grande de datos

Criterio C: Completitud. Se prefiere una solución que pueda almacenar los datos necesarios para la solución

[3] Puede guardar todos los datos del pasajero

[2] Puede buscar algunos datos del pasajero

[1] Puede almacenar solo 1 dato del pasajero.

Criterio D: Familiaridad con la solución. Se prefiere una solución con la que el equipo sea familiar. La solución puede ser:

[3] Familiar

­[2] Conocida

­[1] No conocida

|  | Criterio A | Criterio B | Criterio C | Criterio D | Total |
| --- | --- | --- | --- | --- | --- |
| HashTable | Constante  4 | Moderada  3 | Todos los datos  3 | Familiar  3 | 13 |
| HashSet | Constante  4 | Casi no pierde  4 | Todos los datos  3 | No conocida  1 | 12 |

Entrada y salida de pasajeros:

Criterio A: Eficiencia. Se prefiere una solución con mejor eficiencia al insertar y eliminar datos que las otras consideradas. La eficiencia puede ser:

[4] Constante

­[3] Mayor a constante

­[2] Logarítmica

­[1] Lineal

Criterio B: Completitud. Se prefiere una solución que pueda ordenar a los pasajeros tomando en cuenta todos los criterios. La solución puede

[3] Tener en cuenta todos los criterios

[2] Tener en cuenta algunos criterios

[1] Tener en cuenta sólo 1 criterio

Criterio C: Duplicidad. Se prefiere una solución que soporte ordenar datos repetidos

La solución puede:

[2] Soportar datos repetidos

[1] No soportar datos repetidos

|  | Criterio A | Criterio B | Criterio C | Total |
| --- | --- | --- | --- | --- |
| PriorityQueue | Logarítmica  2 | Todos  3 | Soporta  2 | 7 |
| TreeMap | Logarítmica  2 | Todos  3 | No soporta  1 | 6 |

Selección:

De acuerdo con la evaluación anterior se debe seleccionar:

Scanner para lectura de archivos,Gson para deserializacion de datos, HashTable para búsqueda de pasajeros y PriorityQueue para Ingreso y salida de pasajeros, ya que obtuvieron la mayor puntuación en su categoría