

Objetivos

Unidad 4: Algoritmos de Ordenamiento y Búsqueda

Al finalizar esta unidad, el estudiante estará en capacidad de:

- OE4.1 Implementar algoritmos clásicos de ordenamiento de datos en estructuras de datos lineales y aplicarlos en la solución de un problema.
- OE4.2 Implementar algoritmos clásicos de búsqueda de información en estructuras de datos lineales y aplicarlos en la solución de un problema.
- OE4.3 Reconocer la diferencia entre orden natural y orden parcial de los objetos por medio de la descripción de utilidades de las interfaces Comparable y Comparator.
- OE4.4 Calcular el tiempo de ejecución de un algoritmo por medio de las operaciones de tiempo del sistema
- OE4.5 Implementar métodos que permitan generar muestras con datos aleatorios.

Preparación

- Lea cuidadosamente el enunciado, la documentación suministrada y cada uno de los puntos que debe desarrollar antes de empezar su desarrollo. Pregunte a su profesor cualquier duda respecto al enunciado o a los requerimientos funcionales que debe desarrollar.
- Lea cuidadosamente la rúbrica del laboratorio de la unidad 4 (ver rúbrica).
- El trabajo debe ser realizado individualmente.
- El trabajo será entregado en la fecha y hora establecida en Moodle.
- Usted debe utilizar git para manejar el versionamiento de su trabajo desarrollado localmente y manejar como repositorio remoto **privado**, un repositorio en GitHub o un proyecto GitLab. Su trabajo debe ser gestionado con git desde el inicio del desarrollo, los commit deben hacerse regularmente, así como los push al remoto. Esto se verificará con las fechas de los commits. En el momento de la fecha de entrega máxima, usted debe hacer público su repositorio. Recuerde las convenciones de nombre y estructura de directorios indicadas en esta [presentación de Git](#).

Enunciado

PANTALLA DE AEROPUERTO



TIME	AIRLINE	FLIGHT	TO	TERM.	CHK-IN	GATE	EXPE.	REMARKS
14:45	Avianca	AV 080	CARACAS	①		7		GO TO GATE
15:25	Avianca	AV 284	NEW YORK	①		5		GO TO GATE
15:50	Avianca	MM 110	CURACAO	①		4		GO TO GATE
15:50	Avianca	MM 113	GUAYAQUIL	①		7		GO TO GATE
16:10	TACA	LR 690	SAN JOSE C.R.	①	76-80			
16:16	Avianca	CM 302	PANAMA	①	57-60			
16:50	Avianca	AV 067	QUITO	①		1		
17:10	TACA	TA 133	LIMA	①	71-75			
17:58	Avianca	IB 6740	MADRID	①	23-30	18:30		DELAYED
18:00	Avianca	P5 622	PANAMA	①	53-56			

CERRADO: LETICIA

Imagen 1. Pantalla de un terminal aéreo

Con la aparición de las aerolíneas de bajo costo son cada vez más las personas que viajan en avión en el país. Durante 2018 viajaron 37 millones de personas en avión en Colombia, un incremento del 6,1 respecto de 2017¹. Como parte de toda la operación logística para que tantas personas puedan movilizarse a través de las terminales aéreas (aeropuertos), es fundamental la información que en ellos se da de los vuelos a través de todas las pantallas dispuestas para tal fin. En la imagen 1 se muestra la pantalla de un terminal aéreo.

Usted ha sido contratado para simular una pantalla similar que permita mostrar la información de los vuelos pero con hora no militar, pues las personas se han quejado que no entienden muy bien el formato de hora militar. Por tanto usted debe mostrar la hora en formato AM/PM. Para la simulación su programa debe generar aleatoriamente un listado de vuelos en diferentes fechas, horarios, diferentes aerolíneas, diferentes números

¹ <https://www.elcolombiano.com/colombia/cuantas-personas-viajan-en-avion-en-colombia-DH10062818>

Algoritmos de Ordenamiento y Búsqueda

de vuelo (éste debe ser único), ciudades destino y puertas de embarque. El orden por defecto en que los vuelos son mostrados es la hora de salida de menor a mayor.

Una mejora que introducirá su desarrollo de la nueva pantalla de información de vuelos, es que será interactiva. Hasta ahora, las pantallas solo despliegan información pero no permiten que el usuario interactúe con ella, busque su vuelo o reordene los vuelos de acuerdo a diferentes criterios. Su programa permitirá esta interacción y será posible porque ahora las pantallas estarán físicamente al alcance de los usuarios.

Ya que es una simulación, el programa debe permitir cada vez que el usuario lo desee, generar aleatoriamente una nueva lista de vuelos. Debe tener la posibilidad de decidir cuántos vuelos se van a generar, aunque en pantalla el número será limitado, por lo que la vista podrá ser paginada, es decir, el programa despliega los primeros n vuelos (n es la cantidad máxima de vuelos a desplegar en pantalla, pueden ser 20, por ejemplo), y tendrá la posibilidad a través de algún control (botones por ejemplo) de mostrar los siguientes n vuelos y así hasta llegar a los últimos. También debería tener la posibilidad de mostrar los n vuelos anteriores. Es decir, la navegación entre páginas de n vuelos podrá ser hacia adelante y hacia atrás.

Por defecto, el ordenamiento de los vuelos es por fecha y hora, pero la pantalla ofrecerá al usuario la posibilidad de ordenar los vuelos por cualquiera de los otros criterios. También ofrecerá la posibilidad de buscar un vuelo por cualquiera de los criterios, es decir, se pueden buscar vuelos por fecha, por hora, por número de vuelo, por ciudad, etc. Si hay más de un vuelo que concuerde con el criterio buscado, entonces se mostrará el primer vuelo encontrado. Cada vez que se hace una búsqueda o un ordenamiento, el programa debe mostrar el tiempo que se tardó en hacer dicha operación.

En su programa usted deberá:

1. Implementar y utilizar los tres métodos de ordenamiento clásicos: burbuja, selección e inserción.
2. Implementar y utilizar las dos estrategias de búsqueda clásicas: secuencial y binaria.
3. Utilizar la interface Comparable.
4. Utilizar la interface Comparator.
5. Utilizar el método de ordenamiento de la clase Arrays utilizando:
 - a. Comparable.
 - b. Comparator.

Ya que en el programa hay que hacer diferentes ordenamientos y búsquedas, para cumplir con los requisitos anteriores, usted utilizará un algoritmo de ordenamiento en un caso, otro algoritmo en otro caso, el método de ordenamiento de Arrays con Comparable en otro caso y el ordenamiento de Arrays con Comparator en otro caso. Lo mismo para las búsquedas. La interfaz con el usuario debe ser gráfica utilizando JavaFX y debe permitir el cumplimiento de todos los requerimientos descritos. Usted debe proponer la forma concreta que tendrá la interfaz con el usuario, es decir, los componentes, controles, contenedores y su disposición en la misma. Las fechas deberán ser representadas como cadenas de la forma "AAAA-MM-DD" y las horas como objetos compuestos de hora, minutos y momento del día (este último con valores posibles AM y PM).

Entregables.

1. Requerimientos Funcionales.
2. Diagrama de clases de modelo y control de la interfaz (no generado automáticamente)
3. Implementación completa de todos los requerimientos en Java.

Algoritmos de Ordenamiento y Búsqueda

4. Tabla de trazabilidad de requerimientos vs métodos (tabla con una columna de los requerimientos, tal que, por cada requerimiento se indica en la columna siguiente todos los métodos que contribuyen a resolverlo).

Fecha de Entrega Máxima: Martes 9 de Abril de 2019 a las 11:59 AM a través de Moodle. El laboratorio debe trabajarse y entregarse **individualmente**. Lo que usted debe entregar de su trabajo es la url de su repositorio en GitHub o proyecto en GitLab. Recuerde que el repositorio o proyecto debe ser privado durante el desarrollo del laboratorio y hacerse público solo en el momento justo de la entrega máxima indicada aquí.

DOCUMENTACIÓN

Nombre	<u>R.1# Mostrar en pantalla los vuelos.</u>
Resumen	Se mostrará en pantalla cada uno de los vuelos generados de manera aleatoria para la simulación.
Entradas	
Ninguna.	
Resultados	
Vuelos generados de manera automatica.	

Nombre	<u>R.2# Generar una simulación de vuelos de manera aleatoria</u>
Resumen	Se permitirá al usuario generar un número de vuelos en la interfaz
Entradas	
número de vuelos para generar	
Resultados	
vuelos generados de manera aleatoria.	

Nombre	<u>R.3# Permitir organizar los vuelos por criterio</u>
Resumen	Se permitirá organizar todos los vuelos según un criterio en especial
Entradas	
Criterio.	
Resultados	
Vuelos organizados.	

Nombre	<u>R.4# Buscar un vuelo en específico</u>
Resumen	El usuario introducirá según el criterio buscado el vuelo que busca
Entradas	
Criterio, valor de búsqueda,	
Resultados	
El vuelo buscado o en caso de no encontrarse, un aviso.	

Nombre	<u>R.5# Mostrar en pantalla el tiempo que demoró el programa en ejecutar</u>
Resumen	En el programa por cada acción ejecutada se contará que tanto tiempo se demoró el programa en realizar dicha operación.
Entradas	
Ninguna.	
Resultados	
Tiempo de demora.	

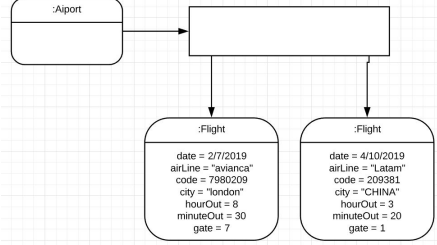
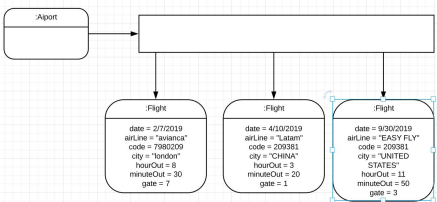
Requerimientos no funcionales

Nombre	<u>RNF.#Trabajar en javafx</u>
Resumen	Se debe generar la interfaz mediante javafx
Entradas	
Ninguna.	
Resultados	
Ninguna.	

Trazabilidad

Clase	Método	Requerimiento
FlightControllerUI Flight AirPort Main	getDate(),getAirLine(),getCode(),getCity(),getHoutOut(), getMinuteOut(),getGate(),getDateOut(),getTime(),getSchedule(),timeToString(),dateOutToString(),getFlights(),generatingFlights(),getFlights"FlightControllerGUI"(),nextAction(),backWard(),	R#1
AirPort FlightControllerGUI	randomAirLine(),randomCode(),randomCity(),randomHour(),randomMinute(),generatinFlights(),getFlights(),	R#2
AirPort FlightControllerGUI	sortDate(), sortTime(), sortAirLine(), sortGate(), sortCode(),sortCity(), SortingAirLine(), SortingCode(), SortingCity(), SortingDate(), SortingGate(), SortingSchedule(),	R#3
AirPort FlightControllerGUI	searchingAirLine(), searchingCity(), searchingCode(), searchingDate(), searchingGates(), searchingSchedule()	R#4
FlightControllerGUI	AlertTime(),realTime()	R#5

Configuración de los casos de prueba

NOMBRE	CLASE	ESCENARIO
setUpScenary1()	AirPortTest	vacío
setupScenary2()	AirPortTest	
setupScenary3()	AirPortTest	

Objetos de la prueba: Comprobar que el programa busca correctamente un vuelo generado por fecha.

Clase	Método	Escenario	Valor de entrada	Resultado
AirPort	searchingDateTest()	setUpScenary2	Date(2/7/2019)	'el programa encontró correctamente el dato buscado'

Objetos de la prueba:Comprobar que el programa busca correctamente un aerolínea buscada.

Clase	Método	Escenario	Valor de entrada	Resultado
AirPort	searchingAirLineTest()	setUpScenary2	searchingAirLine(String 'avianca')	'el programa encontro correctamente el dato buscado'

Objetos de la prueba: Comprobar que el programa busca correctamente mediante el criterio de ciudad

Clase	Método	Escenario	Valor de entrada	Resultado
AirPort	searchingCityTest()	setupScenay2()	searchingCity(String x = "CHINA")	"el programa encontró correctamente el objeto buscado mediante el criterio de ciudad"

Objetos de la prueba: Comprobar que el programa busca correctamente mediante el criterio de hora de salida

Clase	Método	Escenario	Valor de entrada	Resultado
AirPort	searchingHourTest()	setupScenay2()	searchingHour(int x = 3)	"el programa encontró correctamente el vuelo deseado mediante el criterio de hora de salida"

Objetos de la prueba: Comprobar que el programa ordena correctamente mediante el criterio de fecha de salida de menor a mayor

Clase	Método	Escenario	Valor de entrada	Resultado
AirPort	sortDateTest()	setupScenary3	ninguno	"el programa ordeno correctamente cada uno de los vuelos"

Objetos de la prueba: Comprobar que el programa ordena correctamente mediante el criterio de tiempo de salida de menor a mayor

Clase	Método	Escenario	Valor de entrada	Resultado
AirPort	sortTimeTest()	setupScenary3	ninguno	‘el programa ordeno correctamente cada uno de los vuelos “

Objetos de la prueba: Comprobar que el programa ordena correctamente mediante el criterio de aerolínea de menor a mayor según su orden lexicografico.

Clase	Método	Escenario	Valor de entrada	Resultado
AirPort	sortAirLineTest()	setupScenary3	ninguno	‘el programa ordeno correctamente cada uno de los vuelos “

Objetos de la prueba: Comprobar que el programa ordena correctamente mediante el criterio de puerta de embarque de mayor a meno

Clase	Método	Escenario	Valor de entrada	Resultado
AirPort	sortGateTest()	setupScenary3	ninguno	‘el programa ordeno correctamente cada uno de los vuelos “

Objetos de la prueba: Comprobar que el programa ordena correctamente mediante el criterio de códigos de vuelo de menor a mayor

Clase	Método	Escenario	Valor de entrada	Resultado
AirPort	sortCodeTest()	setupScenary3	ninguno	‘el programa ordeno correctamente cada uno de los vuelos “

Objetos de la prueba: Comprobar que el programa ordena correctamente mediante el criterio de ciudad de menor a mayor según el orden lexicográfico

Clase	Método	Escenario	Valor de entrada	Resultado
AirPort	sortCityTest()	setupScenary3	ninguno	‘el programa ordeno correctamente cada uno de los vuelos “