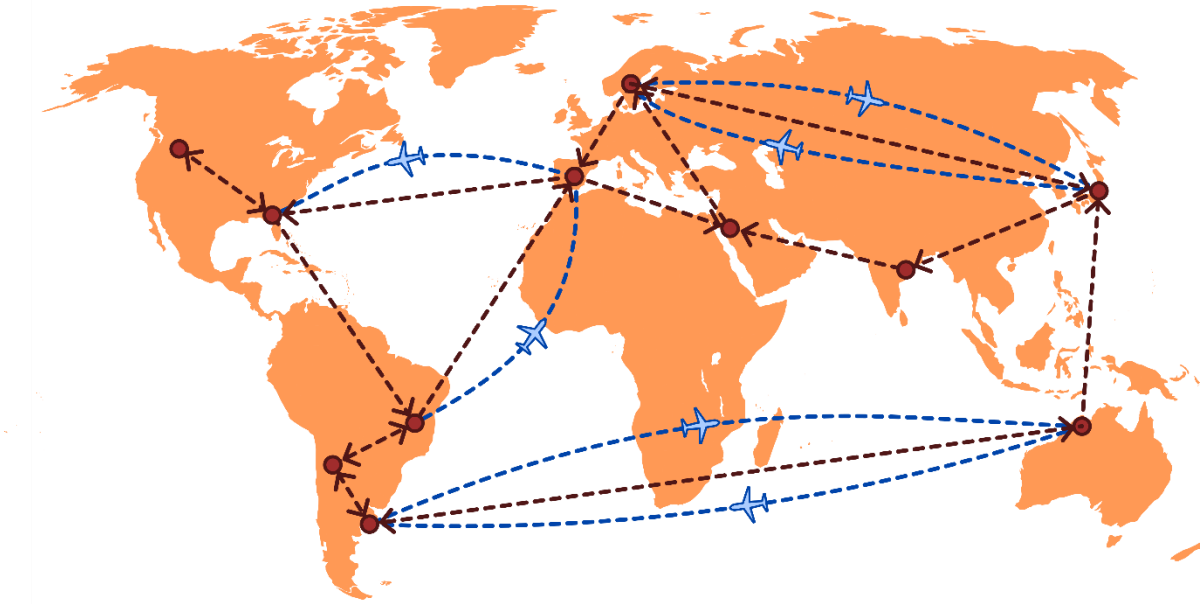
EVALUACION	Obligatorio	**GRUPO**	Todos	**FECHA**	Marzo 2022
MATERIA	Algoritmos y Estructuras de Datos 2				
CARRERA	Analista Programador – Analista en TI				

CONDICIONES	• Puntos: Máximo: 40 Mínimo: 1
	• Fecha máxima de entrega: **02/06/2022**
	• LA ENTREGA SE REALIZA EN FORMA ONLINE EN ARCHIVO NO MAYOR A 40MB EN FORMATO ZIP, RAR.
	• **IMPORTANTE:**
	  ○ Inscribirse
	  ○ Formar grupos de hasta 3 personas.
	  ○ Subir el trabajo a Gestión antes de la hora indicada				

Obligatorio: **Sistema para Viajes**

Introducción

Se desea implementar un programa para resolver la alta demanda de las operaciones de una agencia de viaje cumpliendo con las cotas de tiempo requeridas para cada operación. Se necesita poder ingresar y consultar información sobre los pasajeros, aeropuertos, conexiones y vuelos.



Todas las operaciones deberán devolver una instancia de la clase Retorno. Dicha clase contiene:

- Un **resultado**, que especifica si la operación se pudo realizar correctamente (OK), o si ocurrió algún error (según el número de error).
- Un **valorEntero**, para las operaciones que retornen un número entero.
- Un **valorString**, para las operaciones que retornen un String, o un valor más complejo, por ejemplo, un listado, el cual será formateado según lo indicado en los ejemplos.

```
public class Retorno {
    public enum Resultado {OK, ERROR_1, ERROR_2, ERROR_3, ERROR_4,
NO_IMPLEMENTADA};
    public int valorEntero;
    public String valorString;
    public Resultado resultado;
}
```

Se provee: una interfaz llamada **Sistema**, la cual no podrá ser modificada en ningún sentido, y una clase **SistemaImp** que la implementa, donde su equipo deberá completar la implementación de las operaciones solicitadas.

Consideraciones:

- La clase sistema NO PODRÁ SER UN SINGLETON. Debe ser una clase **instanciable**.
- Pueden definirse tipos de datos (clases) auxiliares.
- Se provee un proyecto base con la estructura de las clases.

Funcionalidades

01. Inicializar Sistema

```
Retorno inicializarSistema (int maxAeropuertos);
```

Descripción: Inicializa las estructuras necesarias para representar el sistema especificado, capaz de registrar como máximo la cantidad `maxAeropuertos` de aeropuertos diferentes en el sistema.

Restricción de eficiencia: no tiene.

Retornos posibles	
OK	El sistema pudo ser inicializado exitosamente.
ERROR	1. Si *maxAeropuertos* es menor o igual a 2.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

02. Registrar pasajero

```
Retorno registrarPasajero(String cedula, String nombre, String
teléfono, Categoria categoria);
```

Descripción: Registra el pasajero con sus datos. La cédula es su identificador único.
Restricción de eficiencia: Esta operación deberá realizarse en orden (log n) promedio.

Retornos posibles	
OK	El pasajero fue registrado exitosamente.
ERROR	1. Si alguno de los parámetros es vacío o *null*. 2. Si la cédula no es una cédula con formato válido. 3. Si ya existe un pasajero registrado con esa cédula.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Restricción: (Investigación) Se requiere el uso de expresiones regulares para lograr validar el formato de la cédula: N.NNN.NNN-N o NNN.NNN-N (No es necesario realizar la validación con el dígito verificador de la cédula)

Las categorías posibles son:

- A: Platino
- B: Frecuente
- C: Estándar
- D: Esporádico
- E: Nuevo

Ejemplos:

Resultado: `OK`

```
registrarPasajero("1.345.345-4", "Guillermo", "555-054888",
Categoria.D);
```

Resultado esperado: `ERROR_2`

```
registrarPasajero("1.3.45.345-4", Guillermo", "555-054888",
Categoria.D);
registrarPasajero("0.345.345-4","Guillermo", "555-054888",
Categoria.D);
```

03. Buscar Pasajero

`Retorno` **`buscarPasajero`**`(String cedula);`

Descripción: Retorna en valorString los datos del pasajero con el formato "`cedula;nombre;telefono;categoria`". Además, en el campo valorEntero de la clase Retorno, deberá devolver la cantidad de elementos que recorrió durante la búsqueda en la estructura utilizada.

Restricción de eficiencia: Esta operación deberá realizarse en orden (log n) promedio.

Retornos posibles	
OK	Si el usuario se encontró. Retorna en *valorString* los datos del pasajero. Retorna en *valorEntero* la cantidad de elementos recorridos durante la búsqueda.
ERROR	1. Si la cédula no tiene formato válido. 2. Si no existe un pasajero registrado con esa cédula.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valorString: `cedula;nombre;teléfono;categoría.`

Por ejemplo, valorString del retorno en una consulta válida:

`1.345.345-4;Guillermo;555-054888;Esporádico`

04. Listar pasajeros por cédula ascendente

```
Retorno listarPasajerosAscendente();
```

Descripción: Retorna en valorString los datos de todos los pasajeros registrados, ordenados por cédula en forma creciente (numéricamente).
Restricción de eficiencia: Esta operación deberá realizarse en orden (n).

Retornos posibles	
OK	Retornando el listado de pasajeros en *valorString*.
ERROR	No hay errores posibles.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valor String:
`cédula1;nombre1;teléfono1;categoría1|cédula2;nombre2;telefono2;categoria2`

Por ejemplo:
`1.345.345-4;Guillermo;555-054888;Esporádico|4.985.345-4;Alberto;555-044488;Frecuente`

05. Listar pasajeros por cédula descendente

```
Retorno listarPasajerosDescendente();
```

Descripción: Retorna en valorString los datos de todos los pasajeros registrados, ordenados por cédula en forma decreciente (numéricamente).
Restricción de eficiencia: Esta operación deberá realizarse en orden (n).

Retornos posibles	
OK	Retornando el listado de pasajeros en *valorString*.
ERROR	No hay errores posibles.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valor String:
`cédula1;nombre1;teléfono1;categoría1|cédula2;nombre2;telefono2;categoria2`

Por ejemplo:
`4.985.345-4;Alberto;555-044488;Frecuente|1.345.345-4;Guillermo;555-054888;Esporádico`

06. Listar pasajeros por categoría

```
Retorno listarPasajerosPorCategoría(Categoria unaCategoria);
```

Descripción: Retorna en valorString los datos de todos los pasajeros registrados con esa categoría. No se requiere que estén ordenados.
Restricción de eficiencia: Esta operación deberá realizarse en orden (k), siendo k la cantidad de pasajeros con dicha categoría.

Retornos posibles	
OK	Se pudo listar los pasajeros que pertenecen a esa categoría correctamente.
ERROR	No hay errores posibles.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valor String:
`cédula1;nombre1;teléfono1;categoría1|cédula2;nombre2;telefono2;categoria2`

Por ejemplo:
`4.985.345-4;Alberto;555-044488;Frecuente|1.345.345-4;Guillermo;555-054888;Frecuente`

07. Registrar aeropuerto

```
Retorno registrarAeropuerto(String codigo, String nombre);
```

Descripción: Registra el aeropuerto en el sistema con el código y nombre indicado. El código es el identificador único, el código y nombre no pueden ser vacíos

Retornos posibles	
OK	Si el aeropuerto fue registrado exitosamente.
ERROR	1. Si en el sistema ya hay registrados *maxAeropuertos.* 2. Si código o nombre son vacíos o *null* 3. Si ya existe un aeropuerto con ese código.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Esta operación no tiene restricciones de eficiencia.

08. Registrar Conexión

```
Retorno registrarConexion(String codigoAeropuertoOrigen, String codigoAeropuertoDestino, double kilometros);
```

Descripción: Registra una conexión en el sistema desde el aeropuerto codigoAeropuertoOrigen al aeropuerto codigoAeropuertoDestino.

Retornos posibles	
OK	Si la conexión fue registrada exitosamente.
ERROR	1. Si *kilometros* es menor o igual a 0. 2. Si no existe el aeropuerto de origen 3. Si no existe el aeropuerto de destino 4. Si ya existe una conexión entre el origen y el destino
NO_IMPLEMENTADA	Cuando aún no se implementó.

Nota: Se considera que las conexiones no son navegables en ambos sentidos. O sea que, si existe una conexión entre los aeropuertos A al B, puede no existir de B al A.

09. Registrar vuelo

```
Retorno registrarVuelo(String codigoAeropuertoOrigen, String codigoAeropuertoDestino,String codigoDeVuelo, double combustible, double minutos, double costoEnDolares);
```

Descripción: Registra un nuevo vuelo en el sistema, debe existir la conexión entre el aeropuerto de origen y el destino. Puede haber varios vuelos entre el aeropuerto de origen y el destino. El identificador de un vuelo es el aeropuertoOrigen, el aeropuertoDestino y el codigoDeVuelo. En cada conexión no puede haber dos vuelos con el mismo código.

Retornos posibles	
OK	Si el vuelo se registró exitosamente.
ERROR	1. Si alguno de los parámetros double es menor o igual a 0. 2. Si alguno de los parámetros String es vacío o *null*. 3. Si no existe el aeropuerto de origen. 4. Si no existe el aeropuerto de destino. 5. Si no existe una conexión entre origen y destino 6. Si ya existe un vuelo con ese código en esa conexión.
NO_IMPLEMENTADA	Cuando aún no se implementó.

10. Actualizar vuelo

```
Retorno actualizarVuelo(String codigoAeropuertoOrigen, String codigoAeropuertoDestino,String codigoDeVuelo, double combustible, double minutos, double costoEnDolares);
```

Descripción: Actualiza un vuelo existente en el sistema, debe existir la conexión entre el aeropuerto de origen y el destino. Pueden haber varios vuelos entre el aeropuerto de origen y el destino. El identificador de un vuelo es el codigoAeropuertoOrigen, el codigoAeropuertoDestino y el codigoDeVuelo. En cada conexión no puede haber dos vuelos con el mismo código.

Retornos posibles	
OK	Si el vuelo se registro exitosamente.
ERROR	1. Si alguno de los parámetros double es menor o igual a 0. 2. Si alguno de los parámetros String es vacío o null. 3. Si no existe el aeropuerto de origen. 4. Si no existe el aeropuerto de destino. 5. Si no existe una conexión entre origen y destino 6. Si no existe un vuelo con ese código en esa conexión.
NO_IMPLEMENTADA	Cuando aún no se implementó.

11. Aeropuertos por Cantidad de escalas

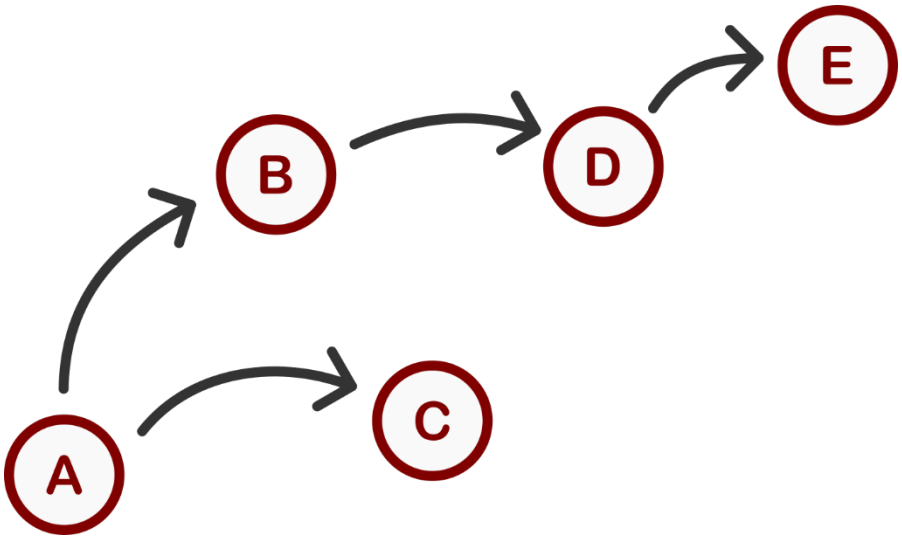
```
Retorno listadoAeropuertosCantDeEscalas(String
codigoAeropuertoDeOrigen, int cantidad);
```

Descripción: Dado un aeropuerto de origen se debe retornar en el valorString los datos de los aeropuertos (ordenados por código creciente) a los que se pueda llegar realizando hasta la cantidad de escalas indicada por parámetro.

Retornos posibles	
OK	Retorna en *valorString* los datos de aeropuertos a los que se pueda llegar con hasta "cantidad" de escalas.
ERROR	1. Si la cantidad es menor que cero. 2. Si el aeropuerto no está registrado en el sistema.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valor String:

```
codigoAeropuerto1;nombreAeropuerto1|
codigoAeropuerto2;nombreAeropuerto2
```



12. Viaje de costo mínimo en kilometros

```
Retorno viajeCostoMinimoKilometros(String codigoAeropuertoOrigen, String codigoAeropuertoDestino);
```

Descripción: Retorna el camino más corto en kilómetros que podría realizar un pasajero para ir del aeropuerto de origen al de destino.

Retornos posibles	
OK	Si el camino pudo ser calculado exitosamente. Retorna en *valorEntero* la cantidad de kilometros total del camino. Retorna en *valorString* el camino desde el aeropuerto de origen al de destino incluidos.
ERROR	1. Si alguno de los códigos es vacío o *null*. 2. Si no hay camino entre el origen y el destino.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valor String:

```
<codigoAeropuertoOrigen>;<nombreAeropuertoOrigen|<codigoAeropuerto1>;<nombreAeropuerto1|<codigoAeropuerto2>;<nombreAeropuerto2|<codigoAeropuertoDestino>;<nombreAeropuertoDestino
```

13. Viaje de costo mínimo en dólares

```
Retorno viajeCostoMinimoDolares(String codigoAeropuertoOrigen, String codigoAeropuertoDestino);
```

Descripción: Retorna el camino menos costoso en dolares que podría realizar un pasajero para ir del aeropuerto de origen al de destino.

Retornos posibles	
OK	Si el camino pudo ser calculado exitosamente. Retorna en *valorEntero* la cantidad de dólares total del camino. Retorna en *valorString* el camino desde el aeropuerto de origen al de destino incluidos.
ERROR	1. Si alguno de los códigos es vacío o *null*. 2. Si no hay camino entre el origen y el destino.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valor String:

```
<codigoAeropuertoOrigen>;<nombreAeropuertoOrigen|<codigoAeropuerto1>;<nombreAeropuerto1|<codigoAeropuerto2>;<nombreAeropuerto2|<codigoAeropuertoDestino>;<nombreAeropuertoDestino
```

14. Consultas disponibilidad

```
Retorno consultaDisponibilidad(int[][] matriz, int cantidad, Clase unaClase);
```

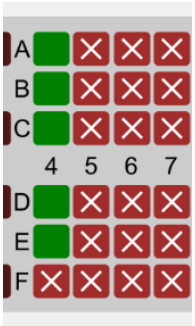
Descripción: Se agregó un módulo para verificar la disponibilidad de asientos en los vuelos, para esto se recibe por parámetros una matriz de int en la que están cargados los pasajeros actuales. La cantidad refiere a la cantidad de asientos contiguos (verticalmente) que se desean, la clase es un enumerado con el que se desea filtrar la clase deseada.
En la matriz:

- 0 es disponible, 1 es ocupado y 2 es no disponible.
- No hay una fila para el pasillo central.
- El tamaño (cantidad de filas y columnas) es siempre el mismo.

Retornos posibles	
OK	Si hay disponibilidad. Retorna en *valorEntero* la cantidad de opciones disponibles. Retorna en *valorString* el listado de las distintas opciones de disponibilidad para la consulta.
ERROR	1. Si la cantidad es menor o igual a 0.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno para una consulta para clase ejecutiva con cantidad 3:

ValorEntero: 3
ValorString: `A4-B4-C4|B4-C4-D4|C4-D4-E4`



Información importante

Se deberán **respetar los formatos de retorno** dados para las operaciones que devuelven datos.

Está **terminantemente prohibido** el uso de clases de Java tales como **ArrayList**.

Ninguna de las operaciones deben imprimir **nada** en consola.

El sistema no debe requerir ningún tipo de interacción con el usuario por consola.

Es obligación del estudiante mantenerse al tanto de las aclaraciones que se realicen en clase o a través del foro de aulas.

Se valorará la selección adecuada de las estructuras para modelar el problema y la eficiencia en cada una de las operaciones. Deberá aplicar la metodología vista en el curso.

El proyecto será implementado en lenguaje JAVA sobre una interfaz Sistema que se publicará en el sitio de la materia en aulas.ort.edu.uy (El uso de esta interfaz es obligatorio).

El proyecto entregado debe compilar y ejecutar correctamente en IntelliJ IDEA.

No se contestarán dudas sobre el obligatorio en las 48 horas previas a la entrega.