

<b>EVALUACION</b>	Obligatorio	<b>GRUPO</b>	Todos	<b>FECHA</b>	Agosto 2023
<b>MATERIA</b>	Algoritmos y Estructuras de Datos 2				
<b>CARRERA</b>	Analista Programador – Analista en TI				
<b>CONDICIONES</b>	<ul style="list-style-type: none"> <li>Puntos: <b>Máximo: 35</b>      Mínimo: 1</li> <li><b>Fecha máxima de entrega: 01/11/2023</b></li> <li>LA ENTREGA SE REALIZA EN FORMA ONLINE EN ARCHIVO NO MAYOR A 40MB EN FORMATO ZIP, RAR.</li> <li><b>IMPORTANTE:</b> <ul style="list-style-type: none"> <li>Inscribirse</li> <li>Formar grupos de <b>hasta 2 personas</b>.</li> <li>Subir el trabajo a Gestión antes de la hora indicada, ver hoja al final del documento: "RECORDATORIO"</li> </ul> </li> </ul>				

## Introducción

Se desea implementar un software para la gestión y recomendación de viajeros, al gestionar una gran cantidad de ciudades, conexiones y viajeros debemos ser cuidadosos a la hora de diseñar las estructuras.

Luego de un breve análisis, se creó una interfaz para separar las reglas de negocio de la parte del manejo de servidor donde 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, ERROR_5, ERROR_6, ERROR_7, NO_IMPLEMENTADA};
    public int valorEntero;
    public String valorString;
    public Resultado resultado;
}
```

Además, se provee: una interfaz llamada **Sistema**, la cual no podrá ser modificada en ningún sentido, y una clase **ImplementacionSistema** 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, pero por ningún motivo puede modificarse las firmas definidas en la interfaz Sistema.
- Se provee un proyecto base con la estructura base de las clases, utilizar este proyecto es obligatorio.

## Funcionalidades

### 01. Inicializar Sistema

Retorno **inicializarSistema** (int maxCiudades);

Descripción: Inicializa las estructuras necesarias para representar el sistema especificado, capaz de registrar como máximo la cantidad `maxCiudades` de ciudades.

Restricción de eficiencia: no tiene.

Retornos posibles	
OK	Si el sistema pudo ser inicializado exitosamente.
ERROR	1. Si <code>maxCiudades</code> es menor o igual a 5.
NO_IMPLEMENTADA	Cuando aún no se implementó. Es el tipo de retorno por defecto.

## 02. Registrar viajero

Retorno **registrarViajero**(String cedula, String nombre, int edad, TipoViajero tipo);

**Descripción:** Registra el viajero con sus datos. Para simplificar el problema se asume que todos los viajeros tienen una cédula de identidad única.

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

Retornos posibles	
OK	El viajero fue registrado exitosamente.
ERROR	<ol style="list-style-type: none"> <li>1. Si alguno de los parámetros es vacío o <i>null</i>.</li> <li>2. Si la cédula no tiene el formato válido.</li> <li>3. Si ya existe un viajero registrado con esa cédula.</li> </ol>
NO_IMPLEMENTADA	Cuando aún no se implementó.

**Validación de cédula:** Se requiere el uso de expresiones regulares para lograr validar el formato de la cédula: N.NNN.NNN-N o NNN.NNN-N (No se debe realizar la validación con el dígito verificador de la cédula)

Los posibles tipos de viajero son:

- 1: Premium
- 2: Estándar
- 3: Casual

Ejemplos:

**Resultado:** OK

```
registrarViajero("3.345.345-4", "Fabiana", 12, TipoViajero.PREMIUM);
```

**Resultado esperado:** ERROR\_2

```
registrarViajero("1.3.45.345-4", "Marcos", 9, TipoViajero.ESTANDAR);
```

```
registrarViajero ("0.345.345-4", "Ana", 40, TipoViajero.CASUAL);
```

### 03. Buscar Viajero

Retorno **buscarViajero**(String cedula);

**Descripción:** Retorna en valorString los datos del viajero según el formato indicado. 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  $O(\log n)$  promedio.

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

Formato de retorno del valorString: cédula;nombre;edad;tipo

Ejemplo; 3.456.234-2;Ana;32;Casual

### 04. Listar viajeros por cédula descendente

Retorno **listarViajerosDescendente**();

**Descripción:** Retorna en valorString los datos de todos los viajeros registrados, ordenados **numéricamente** por cédula en forma descendente.

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

Retornos posibles	
OK	Retornando el listado de viajeros en <i>valorString</i> . En caso de no haber viajeros, <i>valorString</i> será un String vacío.
ERROR	No hay errores posibles.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del valorString con *cedula1* > *cedula2*:

*cedula1*;nombre1;edad1;tipo1|*cedula2*;nombre2;edad2;tipo2

Ejemplo: 1.234.233-2;Alberto;20;Casual|234.233-3;Ana;32;Casual

## 05. Listar viajeros por cédula ascendente

Retorno `listarViajerosAscendente()` ;

**Descripción:** Retorna en `valorString` los datos de todos los viajeros registrados, ordenados **numéricamente** por cédula en forma ascendente.

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

Retornos posibles	
OK	Retornando el listado de viajeros en <i>valorString</i> . En caso de no haber viajeros, <i>valorString</i> será un String vacío.
ERROR	No hay errores posibles.
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del `valorString` con `cedula1 < cedula2`  
`cedula1;nombre1;edad1;tipo1|cedula2;nombre2;edad2;tipo2`

Ejemplo: `234.233-2;Ana;32;Casual|1.234.233;Pedro;20;Premium`

## 06. Listar viajeros por tipo

Retorno `listarViajerosPorTipo(TipoViajero tipo)` ;

**Descripción:** Retorna en `valorString` los datos de todos los viajeros registrados con ese tipo. **Se requiere que estén ordenados de forma creciente.**

**Restricción de eficiencia:** Esta operación deberá realizarse en orden (k), siendo k la cantidad de viajeros con dicho tipo.

Retornos posibles	
OK	Devuelve en <code>valorString</code> el listado de viajeros de ese tipo o un String vacío si no hay viajeros de ese tipo.
ERROR	1. Si el tipo es null
NO_IMPLEMENTADA	Cuando aún no se implementó.

Formato de retorno del `valorString`: **con `cedula1 < cedula2`**

`cedula1;nombre1;edad1;tipo1|cedula2;nombre2;edad2;tipo2`

## 07. Registrar ciudad

Retorno **registrarCiudad**(String codigo, String nombre);

Descripción: Registra la ciudad en el sistema con el código y nombre indicado. El código es el identificador único alfanumérico formado por dígitos, letras en mayúscula y al menos debe tener largo 5.

Restricción de eficiencia: no tiene.

Retornos posibles	
OK	Si la estación fue registrada exitosamente.
ERROR	<ol style="list-style-type: none"> <li>1. Si en el sistema ya hay registrados <i>maxCiudades</i>.</li> <li>2. Si algún dato es vacío o nulo.</li> <li>3. Si el código es inválido.</li> <li>4. Si ya existe una ciudad con ese código.</li> </ol>
NO_IMPLEMENTADA	Cuando aún no se implementó.

**Resultado:** OK

```
registrarCiudad ("AAQ322", "Paris");
```

**Resultado esperado:** ERROR\_3

```
registrarCiudad ("aaa322", "Paris");
```

```
registrarCiudad ("AA", "Paris");
```

## 08. Registrar Conexión

Retorno **registrarConexion**(String codigoCiudadOrigen, String codigoCiudadDestino, int identificadorConexion, double costo, double tiempo, TipoConexion tipo);

Descripción: Registra una conexión en el sistema desde la ciudad de origen al destino.

Restricción de eficiencia: no tiene.

Retornos posibles	
OK	Si el camino fue registrado exitosamente.
ERROR	<ol style="list-style-type: none"> <li>1. Si alguno de los parámetros double es menor o igual a 0.</li> <li>2. Si alguno de los parámetros String o enum es vacío o null.</li> <li>3. Si alguno de los códigos de ciudad no es válido.</li> <li>4. Si no existe el origen.</li> <li>5. Si no existe el destino.</li> <li>6. Si ya existe una conexión entre el origen y el destino, con ese identificador.</li> </ol>
NO_IMPLEMENTADA	Cuando aún no se implementó.

**Nota:** Se considera que las conexiones no son navegables en ambos sentidos. O sea que, si existe un camino de la ciudad A a la B, puede no existir conexión de B a A. Además, se considera que entre dos ciudades puede haber más de una conexión que te lleva de A a B (se asume que estas se distinguen por su identificador que es un número cualquiera, único para la conexión de A a B).

Los posibles tipos son:

- Ruta Nacional
- Ruta Marítima
- Ruta Aérea
- Ruta Ferroviaria
- Otros

## 09. Actualizar conexión

Retorno **actualizarConexion**(String codigoCiudadOrigen, String codigoCiudadDestino, int identificadorConexion, double costo, double tiempo, TipoConexion tipo);

Descripción: Actualiza una conexión existente en el sistema, debe existir una conexión entre la ciudad de origen y destino.

Restricción de eficiencia: no tiene.

Retornos posibles	
OK	Si el camino se actualizó exitosamente.
ERROR	<ol style="list-style-type: none"> <li>1. Si alguno de los parámetros double es menor o igual a 0.</li> <li>2. Si alguno de los parámetros String o enum es vacío o null.</li> <li>3. Si alguno de los códigos de ciudad no es válido.</li> <li>4. Si no existe la ciudad de origen.</li> <li>5. Si no existe la ciudad de destino.</li> <li>6. Si no existe la conexión con identificador entre origen y destino.</li> </ol>
NO_IMPLEMENTADA	Cuando aún no se implementó.

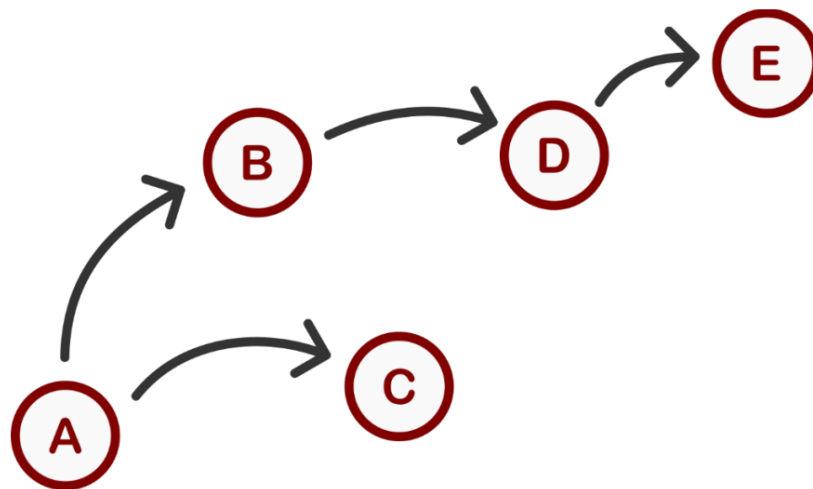


## 10. Ciudades por cantidad de trasbordos

Retorno **listadoCiudadesCantTrasbordos** (String codigo, int cantidad);

Descripción: Dado una ciudad de origen se debe retornar en el valorString los datos de las ciudades (ordenados por código creciente) a los que se pueda llegar realizando hasta la cantidad de trasbordos indicada por parámetro.

Restricción de eficiencia: no tiene.



Retornos posibles	
OK	Retorna en <i>valorString</i> los datos de las ciudades a los que se pueda llegar con hasta "cantidad" de transbordos. Si es cero la cantidad de transbordos debe devolver solo la estación de origen.
ERROR	1. Si la cantidad es menor que cero. 2. Si el código es nulo. 3. Si el código de la ciudad no es válido. 4. Si la ciudad no está registrada en el sistema.
NO_IMPLEMENTADA	Cuando aún no se implementó.

**Formato de retorno del valor String:** con codigoCiudad1 < codigoCiudad2

codigoCiudad1;nombreCiudad1| codigoCiudad2;nombreCiudad2

## 11. Viaje de costo mínimo en tiempo

Retorno **viajeCostoMinimo**(String codigoCiudadOrigen, String codigoCiudadDestino);

Descripción: Retorna el camino más corto (de menor tiempo) que podría realizar un viajero para ir de la ciudad de origen a la de destino. Tener en cuenta que en este viaje se puede pasar por varias ciudades intermedias.

Restricción de eficiencia: no tiene.

Retornos posibles	
OK	Si el camino pudo ser calculado exitosamente. Retorna en <i>valorEntero</i> la cantidad de costo/tiempo total Retorna en <i>valorString</i> el camino desde la ciudad de origen a la de destino incluidas dichas ciudades.
ERROR	1. Si alguno de los códigos es vacío o <i>null</i> . 2. Si alguno de los códigos de ciudad no es válido. 3. Si no hay camino entre el origen y el destino. 4. Si no existe origen. 5. Si no existe destino.
NO_IMPLEMENTADA	Cuando aún no se implementó.

### Formato de retorno del valor String:

```
codigoCiudadOrigen;nombreCiudadOrigen|TIPO_CONEXION|codigoCiudad1;nombreCiudad1|TIPO_CONEXION|codigoCiudad2;nombreCiudad2|TIPO_CONEXION|codigoCiudadDestino;nombreCiudadDestino
```

## Información importante


- Se deberán respetar los formatos de retorno dados para las operaciones que devuelven datos.
- El objetivo de este obligatorio es 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.
- Está terminantemente prohibido el uso de clases de Java tales como ArrayList. La utilización de estas clases implica la pérdida de todos los puntos del obligatorio.
- Los objetos deben se deben comparar utilizando equals.
- Se debe aplicar el uso de buenas prácticas vistas durante la carrera.
- Ninguna de las operaciones implementadas debe imprimir 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.
- 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](http://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.

## RECORDATORIO: IMPORTANTE PARA LA ENTREGA

- **Obligatorios**

La entrega de los obligatorios será en formato digital online, a excepción de algunas materias que se entregarán en Bedelía y en ese caso recibirá información específica en el dictado de la misma.

Los principales aspectos para destacar sobre la **entrega online de obligatorios** son:

1. Ingresá al sistema de Gestión.
2. En el menú, seleccioná el ítem “Evaluaciones” y la instancia de evaluación correspondiente, que figura bajo el título “Inscripto”.
3. Para iniciar la entrega hacé clic en el ícono: 
4. Ingresá el número de estudiante de cada uno de los integrantes y hacé clic en “Agregar”. El sistema confirmará que los integrantes estén inscriptos al obligatorio y, de ser así, mostrará el nombre y la fotografía de cada uno de ellos. Una vez agregados todos los integrantes, hacé clic en “Crear equipo”.

**Cualquier integrante podrá:**

- **Modificar la integración del equipo.**
  - **Subir el archivo de la entrega.**
5. Seleccioná el archivo que deseás entregar. Verificá el nombre del archivo que aparecerá en la pantalla y hacé clic en “Subir” para iniciar la entrega. Cada equipo (hasta 2 estudiantes) debe entregar **un único archivo en formato zip o rar** (los documentos de texto deben ser pdf, y deben ir dentro del zip o rar). El archivo a subir debe tener **un tamaño máximo de 40mb**  
  
Cuando el archivo quede subido, se mostrará el nombre generado por el sistema (1), el tamaño y la fecha en que fue subido.
  6. El sistema enviará un e-mail a todos los integrantes del equipo informando los detalles del archivo entregado y confirmando que la entrega fue realizada correctamente.
  7. Podés cerrar la pestaña de entrega y continuar utilizando Gestión o salir del sistema.
  8. La **hora tope para subir el archivo será las 21:00** del día fijado para la entrega.
  9. La entrega se podrá realizar desde cualquier lugar (ej. hogar del estudiante, laboratorios de la Universidad, etc).
  10. Aquellos de ustedes que presenten alguna dificultad con su inscripción o tengan inconvenientes técnicos, por favor contactarse con el Coordinador o Coordinación adjunta **antes de las 20:00hs.** del día de la entrega, a través de los mails [gervaz@ort.edu.uy](mailto:gervaz@ort.edu.uy), [alamon@ort.edu.uy](mailto:alamon@ort.edu.uy) y [fernandez\\_ma@ort.edu.uy](mailto:fernandez_ma@ort.edu.uy), o telefónicamente al 29021505 - int 1156 (de 8:00 a 14:00 hs) y 1436 (de 17:30 a 20:00 hs).

Si tuvieras una situación particular de fuerza mayor, debes contactarte con suficiente antelación al plazo de entrega, al Coordinador de Cursos ([gervaz@ort.edu.uy](mailto:gervaz@ort.edu.uy)) o Secretario Docente ([paulos@ort.edu.uy](mailto:paulos@ort.edu.uy)).