**Objetivos**

**Unidad 2: Algoritmos de Ordenamiento, Búsqueda y Persistencia**

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

OE3.3 Hacer persistir el estado del modelo de solución del problema durante la ejecución de un programa y restaurarlo cuando se requiera.

OE3.4 Manipular archivos de texto y utilizarlos para implementar requerimientos del cliente relacionados con persistencia.

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.

OE4.4 Calcular el tiempo de ejecución de un algoritmo por medio de las operaciones de tiempo del sistema

**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.
* 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](https://drive.google.com/open?id=1RmergYAoyEvs-Q8g6gklipoKqeHyRhWNW8N77ewTQDQ).

**Enunciado**

Un inversionista de la ciudad quiere construir un sistema de información para tener diferentes clubes de mascotas a nivel nacional, le ha solicitado que construya un programa para el registro de los clubes, de los dueños y de sus respectivas mascotas. La información que se necesita es la siguiente:

* De los clubes es necesario registrar la identificación del club, el nombre, la fecha de creación y tipo de mascotas.
* De los dueños se debe registrar la identificación del dueño, los nombres, los apellidos, la fecha de nacimiento y que tipo de mascota prefiere.
* Y de las mascotas se quiere registrar la identificación de la mascota, su respectivo nombre, fecha de nacimiento que se conozca, género y tipo de mascota.

Una de las necesidades detectadas, es que la información de los clubes debe ser almacenada para su gestión en un archivo plano y para la información de los dueños y las mascotas se debe usar archivos con serialización. El inversionista espera que existan diversos reportes en el sistema de información, estos son:

* Se debe poder generar listados ordenados de los clubes, los dueños y las mascotas por cualquier criterio de los campos solicitados.
* Listado ordenado de los dueños, según el número de mascotas.
* Listado ordenado de los clubes, según el número de dueños.
* El sistema deberá tener opciones de búsqueda por cualquier criterio de los campos solicitados, una solicitud del inversionista es que se realicen de dos formas (usando la manera tradicional y búsqueda binaria) y adicionalmente quiere poder ver en pantalla la comparación de los tiempos que gasta el sistema en realizar la búsqueda tradicional vs la búsqueda binaria.

Tenga en cuenta que cuando se registra una mascota, está será registrada al dueño que se encuentre actualmente en la pantalla, la aplicación deberá verificar que el dueño exista (este registrado en el sistema). El sistema también deberá considerar las siguientes reglas:

* Para registrar un dueño:
  + No deben existir dos dueños con la misma identificación.
* Para registrar una mascota:
  + Ninguna mascota para el mismo dueño se podrá llamar igual.
* Para eliminar un club, un dueño o una mascota, deberá hacerse por el número de identificación o el nombre.
* Implementar y utilizar los tres métodos de ordenamiento clásicos: burbuja, selección e inserción.
* Implementar y utilizar las dos estrategias de búsqueda clásicas: secuencial y binaria.
* Utilizar la interface Comparable.
* Utilizar la interface Comparator.
* Utilizar el método de ordenamiento de la clase Arrays utilizando:
  + Comparable.
  + Comparator.

Con el objetivo de realizar pruebas aceptables al sistema, se debe generar una cantidad de datos importante y sugerimos utilizar la herramienta web https://www.mockaroo.com que permite generar datos aleatorios de forma personalizada. Se debe generar el máximo de filas posible para hacer buenas pruebas y el formato en que se debe exportar debe ser CSV el cual delimita los archivos de texto con coma por defecto.



Ya que el máximo es 1000, realice pruebas del tiempo de la consulta para archivos de entrada mas grande como 100000, por ejemplo (generando 100 veces 1000 y uniendolos).

**Entregables.**

**1.** Requerimientos funcionales y no funcionales.

**2.** Diagrama de clases de modelo y control de la interfaz (no generado automáticamente).

**3.** Implementación completa de todos los requerimientos en Java.

**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).

**5.** Diseño y pruebas de todas las funcionalidades no triviales.

**Fecha de Entrega Máxima:** la que está definida 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.

|  |  |
| --- | --- |
| RF1 | Registrar Club |
| Descripcion | **Registra un Club dentro del Inversionista** |
| Entrada | **Club** |
| Salida | **-** |

|  |  |
| --- | --- |
| RF2 | Registrar Cliente |
| Descripcion | **Registra un Cliente en un Club, pero no pueden tener el ID repetido** |
| Entrada | **Cliente** |
| Salida | **-** |

|  |  |
| --- | --- |
| RF3 | Registrar Mascota |
| Descripcion | **Registra una Mascota en un Cliente, pero no pueden tener el nombre repetido** |
| Entrada |  |
| Salida | **-** |

|  |  |
| --- | --- |
| RF4 | Ordenar Clubes |
| Descripcion | **Se ordenan los clubes ya sea por ID, fecha de creación, nombre o tipos de mascotas** |
| Entrada | **-** |
| Salida | **-** |

|  |  |
| --- | --- |
| RF5 | Ordenar Clientes |
| Descripcion | **Se ordenan los Clientes ya sea por ID, nombre, apellido, fecha de nacimiento o por el tipo de mascota preferida** |
| Entrada | **-** |
| Salida | **-** |

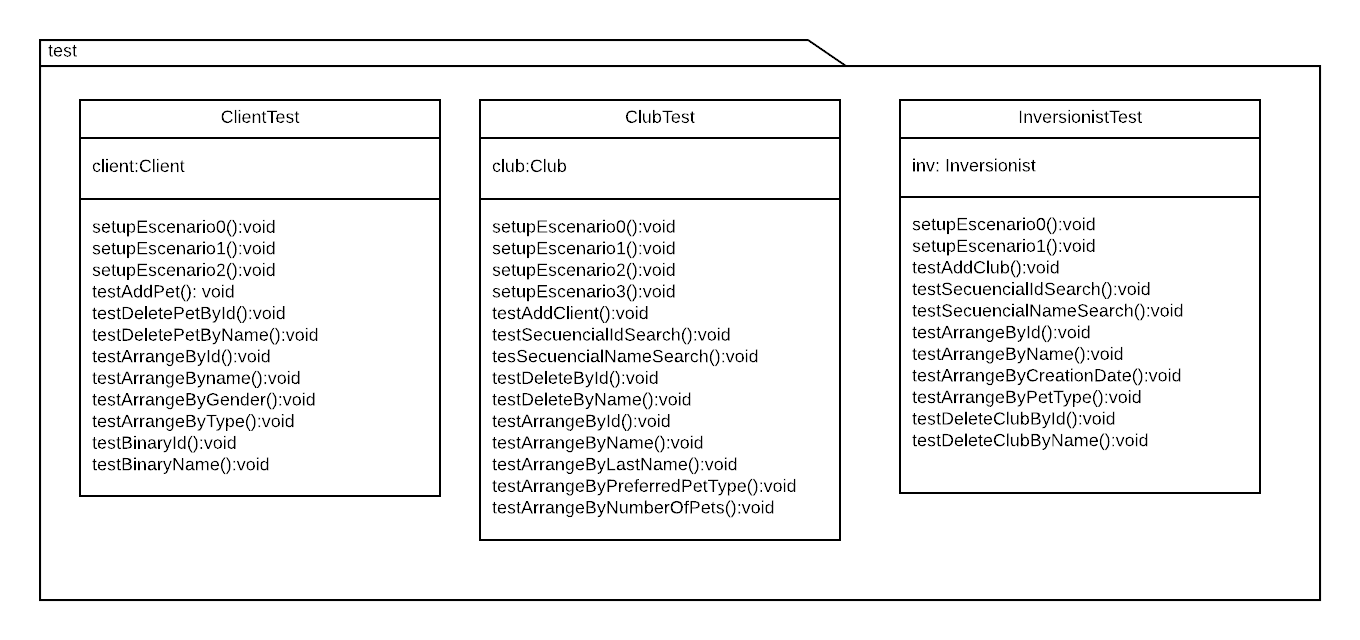
|  |  |
| --- | --- |
| RF6 | Ordenar Mascotas |
| Descripcion | **Se ordenan las Mascotas ya sea por ID, nombre, fecha de nacimiento, genero o tipo de animal** |
| Entrada | **-** |
| Salida | **-** |

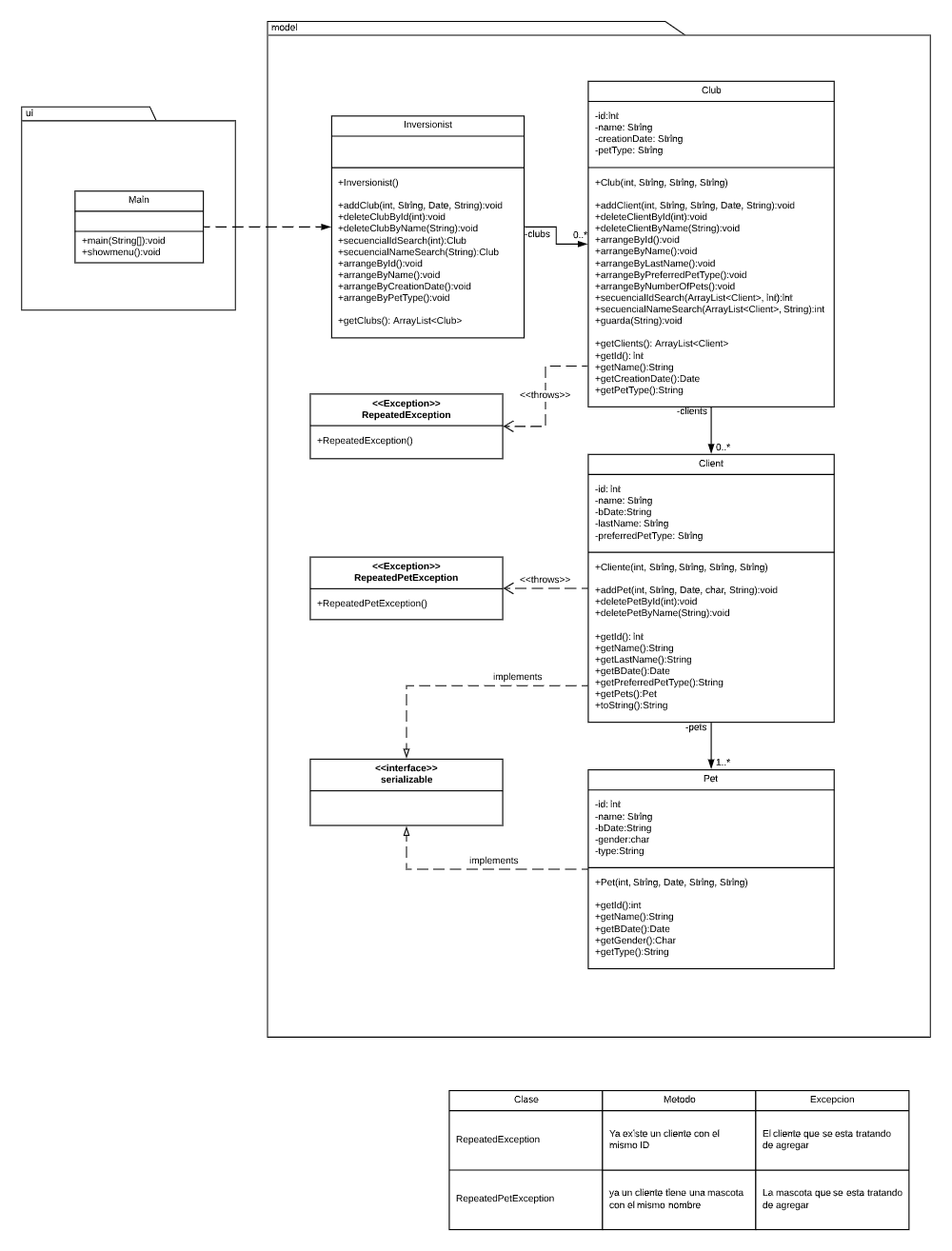
|  |  |
| --- | --- |
| RF7 | Eliminar cliente por ID |
| Descripcion | **Se elimina el cliente que tenga la ID que se ingresa, solo si este existe** |
| Entrada | **Un int que es la ID del cliente a eliminar** |
| Salida | **-** |

|  |  |
| --- | --- |
| RF8 | Eliminar cliente por nombre |
| Descripcion | **Se elimina el cliente que tenga el nombre que se ingresa, solo si este existe** |
| Entrada | **Un String que es el nombre del cliente a eliminar** |
| Salida | **-** |

|  |  |
| --- | --- |
| RF9 | Eliminar Mascota por ID |
| Descripcion | **Se elimina la mascota que tenga el ID que se ingresa, solo si este existe** |
| Entrada | **Un int que es la ID de la mascota a eliminar** |
| Salida | **-** |

|  |  |
| --- | --- |
| RF10 | Eliminar mascota por nombre |
| Descripcion | **Se elimina la Mascota que tenga el nombre que se ingresa, solo si este existe** |
| Entrada | **Un String que es el nombre de la mascota a eliminar** |
| Salida | **-** |



****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clase | Método | Escenario | Entrada | Salida |
| Client | **addPet** | **Hay una mascota que se desea registrar en el ArrayList de un cliente** | **Primero revisa que no haya otra mascota con el mismo nombre y luego, si no esta repetida, lo agrega** | **El cliente ahora tiene una mascota en su ArrayList** |
| **arrangeById** | **Se tienen 3 mascotas en un orden aleatorio dentro de un cliente** | **Realiza un ordenamiento por selección de las mascotas de acuerdo con su ID** | **Las mascotas ahora se encuentran organizadas descendentemente por su ID** |
| **arrangeByName** | **Se tienen 3 mascotas en un orden aleatorio dentro de un cliente** | **Realiza un ordenamiento por selección de las mascotas de acuerdo con su nombre** | **Las mascotas ahora se encuentran organizadas descendentemente por su nombre** |
| **arrangeByBDate** | **Se tienen 3 mascotas en un orden aleatorio dentro de un cliente** | **Realiza un ordenamiento por selección de las mascotas de acuerdo con su fecha de nacimiento** | **Las mascotas ahora se encuentran organizadas descendentemente por su fecha de nacimiento** |
| **arrangeByGender** | **Se tienen 3 mascotas en un orden aleatorio dentro de un cliente** | **Realiza un ordenamiento por selección de las mascotas de acuerdo con su genero** | **Las mascotas ahora se encuentran organizadas descendentemente por su genero** |
| **arrangeByType** | **Se tienen 3 mascotas en un orden aleatorio dentro de un cliente** | **Realiza un ordenamiento por selección de las mascotas de acuerdo con su tipo de animal** | **Las mascotas ahora se encuentran organizadas descendentemente por su tipo** |
| **binaryId** | **Se tienen 3 mascotas en un orden aleatorio dentro de un cliente** | **Realiza una búsqueda binaria dentro de las mascotas para encontrar aquella que tenga el mismo ID que entra como parámetro** | **Retorna una mascota que tiene el ID que se buscaba** |
| **binaryName** | **Se tienen 3 mascotas en un orden aleatorio dentro de un cliente** | **Realiza una búsqueda binaria dentro de las mascotas para encontrar aquella que tenga el mismo nombre que entra como parámetro** | **Retorna una mascota que tiene el nombre que se buscaba** |
| **deletePetById** | **Un cliente tiene una mascota registrada** | **Revisa que dentro de las mascotas que tiene el cliente este la que coincida su ID con la que entra por parámetro y luego la elimina** | **Ahora el cliente no tiene la mascota y por tanto el tamaño de la ArrayList se reduce -1** |
| **deletePetByName** | **Un cliente tiene una mascota registrada** | **Revisa que dentro de las mascotas que tiene el cliente este la que coincida su ID con la que entra por parámetro y luego la elimina** | **Ahora el cliente no tiene la mascota y por tanto el tamaño de la ArrayList se reduce -1** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clase | Método | Escenario | Entrada | Salida |
| Club | **arrangeById** | **Se tienen 3 clientes en un orden aleatorio dentro de un club** | **Se realiza un ordenamiento burbuja de los clientes de acuerdo con su ID** | **Los clientes ahora se encuentran organizados descendentemente por su ID** |
| **arrangeByName** | **Se tienen 3 clientes en un orden aleatorio dentro de un club** | **Se realiza un ordenamiento burbuja de los clientes de acuerdo con su nombre** | **Los clientes ahora se encuentran organizados descendentemente por su nombre** |
| **arrangeByLastName** | **Se tienen 3 clientes en un orden aleatorio dentro de un club** | **Se realiza un ordenamiento burbuja de los clientes de acuerdo con su apellido** | **Los clientes ahora se encuentran organizados descendentemente por su apellido** |
| **arrangeByPreferredPetType** | **Se tienen 3 clientes en un orden aleatorio dentro de un club** | **Se realiza un ordenamiento burbuja de los clientes de acuerdo con su tipo de mascota preferida** | **Los clientes ahora se encuentran organizados descendentemente por su tipo de mascota preferida** |
| **arrangeByNumberOfPets** | **Se tienen 3 clientes en un orden aleatorio dentro de un club** | **Se realiza un ordenamiento burbuja de los clientes de acuerdo con el número de mascotas** | **Los clientes ahora se encuentran organizados ascendentemente por su numero de mascotas** |
| **addClient** | **Se tiene un cliente y se quiere añadir al ArrayList de un Club** | **Se verifica primero que no haya dos clientes con el mismo ID y luego se agrega** | **Ahora el cliente tiene una mascota registrada en su ArrayList** |
| **secuencialIdSearch** | **Se tienen 3 clientes en un orden aleatorio contenidos a su vez en un Club** | **Se realiza una búsqueda secuencial para encontrar la posición del cliente dentro del ArrayList que tenga la misma ID que entra como parámetro** | **Retorna un entero que es la posición del Cliente a buscar** |
| **secuencialNameSearch** | **Se tienen 3 clientes en un orden aleatorio contenidos a su vez en un Club** | **Se realiza una búsqueda secuencial para encontrar la posición del cliente dentro del ArrayList que tenga el mismo nombre que entra como parámetro** | **Retorna un entero que es la posición del Cliente a buscar** |
| **deleteById** | **Hay un club que tiene un cliente registrado** | **Revisa que dentro de los clientes que tiene el club está el que coincida su ID con la que entra por parámetro y luego la elimina** | **Ahora el club no tiene el cliente y por tanto el tamaño de la ArrayList se reduce -1** |
| **deleteByName** | **Hay un club que tiene un cliente registrado** | **Revisa que dentro de los clientes que tiene el club está el que coincida su nombre con la que entra por parámetro y luego la elimina** | **Ahora el club no tiene el cliente y por tanto el tamaño de la ArrayList se reduce -1** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clase | Método | Escenario | Entrada | Salida |
| Inversionist | **addClub** | **Se tiene un club y se quiere añadir al ArrayList del Inversionista** | **Se agrega al ArrayList** | **Ahora el inversionista tiene registrado un club** |
|  | **secuencialIdSearch** | **Se tienen 3 clubes en un orden aleatorio contenidos a su vez en el inversionista** | **Se realiza una búsqueda secuencial para encontrar la posición del club dentro del ArrayList que tenga la misma ID que entra como parámetro** | **Retorna el club que tiene el ID que se buscaba** |
|  | **secuencialNameSearch** | **Se tienen 3 clubes en un orden aleatorio contenidos a su vez en el inversionista** | **Se realiza una búsqueda secuencial para encontrar la posición del club dentro del ArrayList que tenga el mismo nombre que entra como parámetro** | **Retorna el club que tiene el nombre que se buscaba** |
|  | **arrangeById** | **Se tienen 3 clubes en un orden aleatorio contenidos a su vez en el inversionista** | **Se realiza un ordenamiento por inserción de los clientes de acuerdo con su ID** | **Los clubes ahora se encuentran organizados descendentemente por su ID** |
|  | **arrangeByName** | **Se tienen 3 clubes en un orden aleatorio contenidos a su vez en el inversionista** | **Se realiza un ordenamiento por inserción de los clientes de acuerdo con su nombre** | **Los clientes ahora se encuentran organizados descendentemente por su nombre** |
|  | **arrangeByPetType** | **Se tienen 3 clubes en un orden aleatorio contenidos a su vez en el inversionista** | **Se realiza un ordenamiento por inserción de los clientes de acuerdo con su tipo de mascota** | **Los clientes ahora se encuentran organizados descendentemente por su tipo de mascota** |
|  | **deleteClubById** | **El inversionista tiene un club registrado** | **Se busca el club con el ID que entra como parámetro y se elimina** | **Ahora el inversionista no tiene el cliente y por tanto el tamaño de la ArrayList se reduce -1** |
|  | **deleteClubByName** | **El inversionista tiene un club registrado** | **Se busca el club con el nombre que entra como parámetro y se elimina** | **Ahora el inversionista no tiene el cliente y por tanto el tamaño de la ArrayList se reduce -1** |