



Base de datos:

Geovanny Quintero Velez – A00378039

Johan Felipe Jojoa - A00377905

Juan David Garzón - A00377983

Computación y estructuras discretas I

2022

Especificación de requerimientos funcionales

Gen_1: El programa debe ser capaz de generar registros de personas al iniciar el programa. La cantidad de registros la puede indicar el usuario, en caso de no hacerlo se debe generar un número cercano a mil millones de personas (1.000.000.000) o el número más alto que el programa permita. Los datos de los registros abarcaran, un código (autogenerado por el programa), nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad y fotografía.

Cre_1: El programa debe permitir al usuario crear nuevos registros. Este debe digitar todos los campos (datos) que se mencionaron en el requerimiento (Gen_1) menos el código, puesto que este se autogenera. Debe dar la opción de guardar.

Act_1: El programa debe permitir al usuario actualizar la información de los registros de personas. Todos los datos son editables excepto por el código. Debe dar la opción de actualizar los datos (En caso de haber cambios).

Elim_1: El programa debe permitir al usuario eliminar registros. Se borrarán todos los datos asociados al registro de la persona del sistema y del archivo que sería la información (Ser_1). La opción para eliminar debe estar en la interfaz destinada a actualizar (Act_1).

Ser_1: El programa debe permitir la opción de guardar los registros. El guardado de datos debe implementar persistencia, almacenando los registros o los datos de estos en un archivo serializable.

Serch_1: El programa debe permitir buscar a una persona a partir de 4 posibles opciones, las cuales son: por nombre, por apellido, por nombre completo, o por código.

Método de la ingeniería

Identificación del problema

Un proyecto de investigación de la universidad ICESI requiere el desarrollo de un prototipo de software que permita gestionar eficientemente las operaciones CRUD sobre una base de datos de personas de nuestro continente. Se debe simular la creación o generación de registros de personas con los siguientes datos: nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad, fotografía y un código que autogenerará el programa. Adicionalmente, se deben implementar las funciones CRUD (Create, Read, Update y Delete), de modo que sea posible:

- Crear registros, ingresando la información de estos (Exceptuando el código autogenerado).
- Actualizar registros, siendo capaz de editar algunos de sus datos (Exceptuando el código).
- Buscar registros mediante los siguientes campos:
 - Nombre
 - Apellido
 - Nombre y Apellido
 - Código.
- Eliminar registros

Recopilación de información.

- En la sección de búsqueda para los criterios Nombre y Nombre completo debe aparecer una lista emergente de personas de la base de datos cuyos nombres coincidan con los caracteres digitados hasta el momento (Máximo 100 personas)
- En la sección de búsqueda, para cada una de las cuatro búsquedas, se debe mantener un árbol binario de búsqueda auto balanceado o un árbol rojo y negro, que sean persistentes.
- En la sección de búsqueda debe mostrarse al lado del campo donde se digita la cadena, un número que indica la cantidad total de elementos que hasta el momento coinciden con el prefijo digitado.
- En la sección de búsqueda, en el momento en que haya 20 (parametrizable) o menos elementos que coinciden con la búsqueda, debe desplegarse un listado con los registros que coinciden y un botón al lado de cada registro con la opción de Editar, que llevará al formulario con la posibilidad de modificar o eliminar ese registro.
- Se deben implementar pruebas unitarias de las estructuras de datos implementadas y de las operaciones principales de las clases del modelo.
- El programa debe contar con interfaz gráfica.
- Al iniciar se deben preguntar la cantidad de registros que se desean generar, por defecto tiene que ser el mayor valor posible.
- Al generar los registros, el proceso debe contar con una barra de progreso y mostrar cuánto demoró la generación de registros.
- Los datos generados deben poder guardarse en la base de datos del programa mediante una opción.
- Todos los datos del programa deben ser persistentes (es decir, si se cierra el programa, deben seguir allí una vez se inicie nuevamente).
- El programa debe estar en disposición de trabajar con al menos un millón de registros (1.000.000).
- Los datasets para generar los registros se deben guardar en un directorio del proyecto (/data); a continuación, aparecen los datasets junto con la entradas o datos que proporcionan:
 - <https://data.world/alexandra/baby-names> - Nombres
 - <https://data.world/uscensusbureau/frequently-occurring-surnames-from-the-census-2010> - Apellidos
 - https://www.indexmundi.com/es/estados_unidos/distribucion_por_edad.html - Distribución de edades
 - <https://www.kaggle.com/tanuprabhu/population-by-country-2020> - Población por país de América
 - <https://data.world/alexandra/baby-names> - Nombres
- En la generación de registros y en cuanto a la entrada sexo, se puede asumir la misma cantidad de hombres como de mujeres.
- En la generación de registros, la estatura puede escogerse aleatoriamente en un intervalo que tenga sentido.

- En la generación de registros, la fotografía debe ser generada aleatoriamente de este sitio: <https://thispersondoesnotexist.com/> sin importar que su imagen no se corresponda exactamente con su edad, sexo u otros.

Después haber detallado la información del problema, es necesario detallar la información de ciertas estructuras de datos que nos serán útiles para resolver el problema:

Árbol Binario:

Es una estructura de datos en la cual cada nodo puede tener un hijo izquierdo y un hijo derecho. No pueden tener más de dos hijos (de ahí el nombre "binario"). Si algún hijo tiene como referencia a null, es decir que no almacena ningún dato, entonces este es llamado un nodo externo. En el caso contrario el hijo es llamado un nodo interno.

Recuperado de: https://es.wikipedia.org/wiki/%C3%81rbol_binario

Árbol Binario de búsqueda:

Es un árbol binario con la propiedad de que todos los elementos almacenados en el subárbol izquierdo de cualquier nodo x son menores que el elemento almacenado en x , y todos los elementos almacenados en el subárbol derecho de x son mayores que el elemento almacenado en x .

Recuperado de: <https://ccia.ugr.es/~jfv/ed1/c++/cdrom4/paginaWeb/abb.htm>

Véase también: https://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%BAsqueda

Árbol Binario de búsqueda AVL:

Los árboles AVL están siempre equilibrados de tal modo que para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa. Gracias a esta forma de equilibrio (o balanceo), la complejidad de una búsqueda en uno de estos árboles se mantiene siempre en orden de complejidad $O(\log n)$. El factor de equilibrio puede ser almacenado directamente en cada nodo o ser computado a partir de las alturas de los subárboles.

Para conseguir esta propiedad de equilibrio, la inserción y el borrado de los nodos se ha de realizar de una forma especial. Si al realizar una operación de inserción o borrado se rompe la condición de equilibrio, hay que realizar una serie de rotaciones de los nodos.

Recuperado de: https://es.wikipedia.org/wiki/%C3%81rbol_AVL

Árbol rojo y negro:

Un árbol rojo-negro es un árbol binario de búsqueda en el que cada nodo tiene un atributo de color cuyo valor es rojo o negro. En adelante, se dice que un nodo es rojo o negro haciendo referencia a dicho atributo.

Además de los requisitos impuestos a los árboles binarios de búsqueda convencionales, se deben satisfacer las siguientes reglas para tener un árbol rojo-negro válido:

1. Todo nodo es o bien rojo o negro.
2. La raíz es negra.
3. Todas las hojas (NULL) son negras.
4. Todo nodo rojo debe tener dos nodos hijos negros.
5. Cada camino desde un nodo dado a sus hojas descendientes contiene el mismo número de nodos negros.

Recuperado de: https://es.wikipedia.org/wiki/%C3%81rbol_rojo-negro

Búsqueda de soluciones creativas:

Debido a que la solución debía ser implementada por nuestro equipo, se recurrió a los conocimientos que habíamos acumulado durante el semestre, interfaces gráficas en JavaFX y estructuras de datos como el árbol binario, árbol AVL y árbol rojinegro.

Sin embargo, con respecto al requerimiento que pide implementar una lista emergente debajo del campo de búsqueda que se vaya actualizando según los caracteres ingresados, no teníamos idea de cómo desarrollarlo, ya que nuestros conocimientos en JavaFX no estaban tan adelantados, e implementaciones útiles como los hilos no estaban aprendidas aún en nuestro curso de APO II, por lo cual nos encontrábamos bastante limitados. Sin embargo buscando guía en internet, y probando cosas con JavaFX, pensamos en las siguientes alternativas:

Guardado de datos:

Alternativa 1:

Esta alternativa usa el árbol AVL para guardar los datos.

Alternativa 2:

Esta alternativa usa el árbol rojinegro para guardar los datos.

Alternativa 3:

Esta alternativa usa un árbol Binario simple para guardar los datos.

Alternativa 4:

Guardar los datos a comparar de las personas en 4 ArrayList diferentes.

Búsqueda:

Alternativa 1:

Al digitar los caracteres en la barra de búsqueda se toman todos los datos mayores o iguales en el árbol con respecto a los caracteres digitados según el criterio de búsqueda. Cuando queden 100 se muestran en la lista emergente.

Alternativa 2:

Al digitar los caracteres en la barra de búsqueda, tomar todos los datos del árbol y compararlos con los caracteres digitados, los que coincidan agregarlos a la lista emergente.

Alternativa 3:

Mostrar todos los datos en una ventana adicional. ignorando la lista emergente.

Técnica utilizada:

El proceso de generación de posibles ideas se dio principalmente mediante la lluvia de ideas, debido a que espontáneamente surgen formas de implementar una solución. Por tanto se consideró como técnica ideal la lluvia de ideas.

Adicionalmente se tomó un poco de la técnica: lista de atributos, con el fin de facilitar la lluvia de ideas, se separó en 2 la búsqueda: El guardado de datos y la búsqueda de ellos. Aunque la solución es específicamente sobre la búsqueda, esta parte se encuentra fuertemente ligada al cómo se guardan los datos; por ello se consideró pertinente añadirlo aquí.

Transición de la formulación de ideas a diseños preliminares:

Lo primero que hicimos en este paso fue descartar las ideas que no son tan fáciles de implementar y cumplen los mismos requerimientos que otras soluciones. En este sentido descartamos el árbol binario simple y las ArrayList en cuanto al almacenamiento de los datos, debido a que en estos casos los archivos estaban menos ordenados y el tiempo de búsqueda dentro de estos sería mucho mayor probablemente. De igual manera se descartó la alternativa 3, debido a la gran cantidad de datos que mostraría y la poca coincidencia que estos tendrían, obligando al usuario a buscar en una cantidad de datos demasiado grande.

A continuación se detallan más las ideas restantes y se añadieron detalles útiles a las soluciones:

Guardado de datos:

Alternativa 1:

Esta alternativa usa el árbol AVL para guardar los datos. Se aprovecha el balanceo para organizar los datos de manera que se puedan ubicar más fácilmente y con un bajo tiempo de ejecución, haciendo eficiente tanto el guardado como la búsqueda de datos, siendo esta última la que compete a este diseño preliminar. El AVL prima un equilibrio absoluto. Se implementarían 4 árboles, uno por cada criterio de búsqueda.

Alternativa 2:

Esta alternativa usa el árbol rojinegro para guardar los datos. Se aprovecha la menor complejidad en la inserción de los datos para aumentar la eficiencia de estos. Es más complicada la búsqueda de datos debido a que no se organizan siguiendo jerarquías, solo color siguiendo su color. El rojo y negro busca un equilibrio aproximado. Se implementarían 4 árboles, uno por cada criterio de búsqueda.

Búsqueda:

Alternativa 1:

Al digitar los caracteres en la barra de búsqueda se toman todos los datos mayores o iguales en el árbol con respecto a los caracteres digitados según el criterio de búsqueda. Se comparan los datos con los caracteres digitados según el criterio de búsqueda. Cuando queden 100 se muestran en la lista emergente.

Alternativa 2:

Al digitar los caracteres en la barra de búsqueda, tomar todos los datos del árbol y compararlos con los caracteres digitados según el criterio de búsqueda, los que coincidan agregarlos a la lista emergente.

Evaluación y selección de la mejor solución:

Se evaluarán las ideas teniendo en cuenta la siguiente rúbrica:

Requerimientos	Éxito total (3)	Éxito alto (2)	Éxito suficiente (1)	Éxito Nulo (0)	Puntuación
Número de coincidencias mostradas	Muestra un máximo de 100 coincidencias encontradas	Muestra un máximo de alrededor de 100 coincidencias encontradas	Muestra unas pocas coincidencias encontradas	No muestra coincidencias	
Selección de coincidencia	Permite seleccionar la coincidencia requerida para luego editarla, mostrando un botón cuando queden menos de 20 coincidencias encontradas	Permite seleccionar la coincidencia requerida para luego editarla, cuando queden menos de 20 coincidencias encontradas	Permite seleccionar la coincidencia requerida para luego editarla, con cualquier número de coincidencias encontradas, sin botón	No permite seleccionar coincidencias	
Implementación lista emergente	Implementa una lista emergente que muestra sugerencias según se vaya escribiendo en ella	Implementa una lista emergente que muestra sugerencias cada ciertos caracteres ingresados	La lista emergente sólo se actualiza cuando se presiona un botón	No implementa la una lista emergente	
Tiempo de búsqueda	Menor a dos segundos	Menor a cinco segundos	Menor a diez segundos	Mayor a diez segundos	
Criterios de búsqueda	Permite buscar según nombre, apellido, nombre completo y código	Permite buscar según la mayoría de criterios	Solo permite buscar por un criterio	No hay criterios disponibles para búsqueda	

Una vez evaluadas las ideas faltantes, quedaron con la siguiente puntuación:

Alternativa 1 = 15.

Alternativa 2 = 13.

Ya que la alternativa 1 tuvo más puntos, se decidió desarrollar esa.

Guardado de datos:

Opciones de Edificio	Requerimiento 1	Requerimiento 2	Requerimiento 3	Requerimiento 4	Requerimiento 5	Total
Alternativa 1	3	3	3	3	3	15
Alternativa 2	3	3	3	2	2	13

Búsqueda:

Opciones de Edificio	Requerimiento 1	Requerimiento 2	Requerimiento 3	Requerimiento 4	Requerimiento 5	Total
Alternativa 1	3	3	3	3	3	15
Alternativa 2	3	3	3	1	3	13

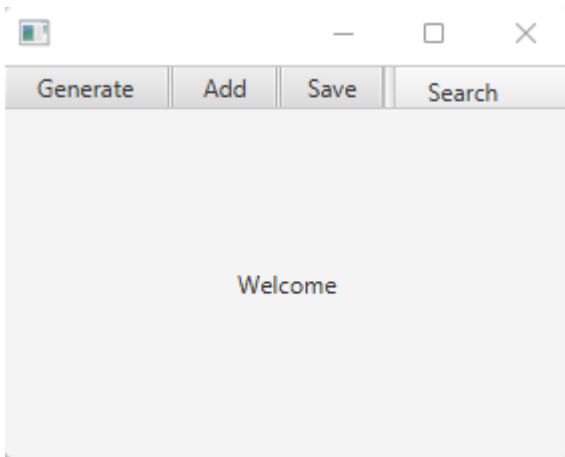
Implementación del diseño:

Con respecto a la estructura en donde se guardarán los datos, se comenzó a implementar los diferentes tipos de árboles en el siguiente orden: árbol binario - árbol AVL - árbol rojinegro.

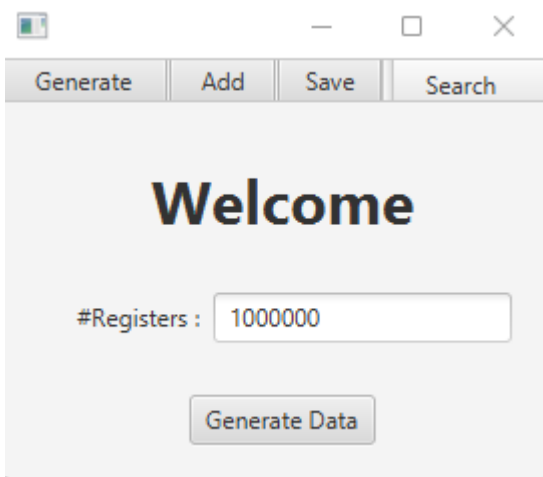
El árbol binario fue implementado con cierta facilidad por parte del equipo, sin embargo, su forma de ordenarse no satisfacía completamente las funcionalidades que queríamos implementar.

Entonces se implementó el árbol AVL, esta implementación se mantuvo hasta los

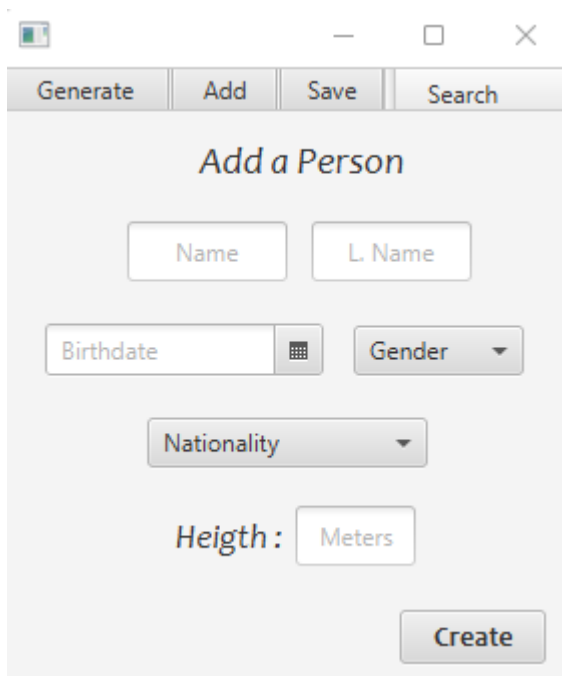
Menú inicial:



Interfaz de generación de archivos en masa:

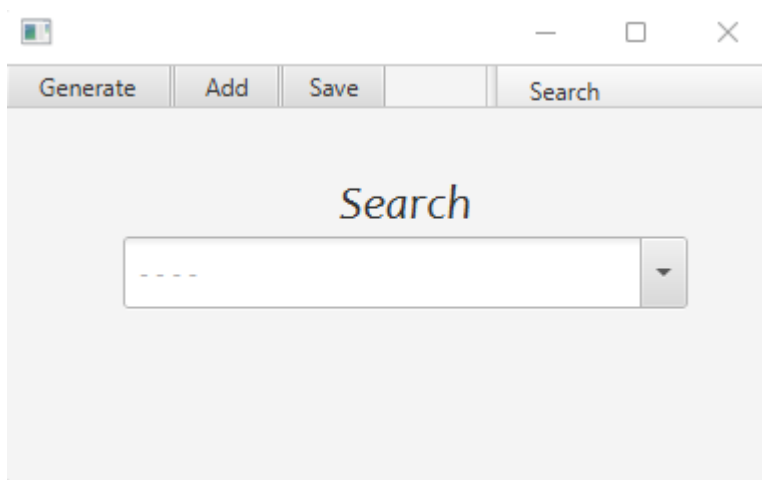


Interfaz de adición manual de registros:



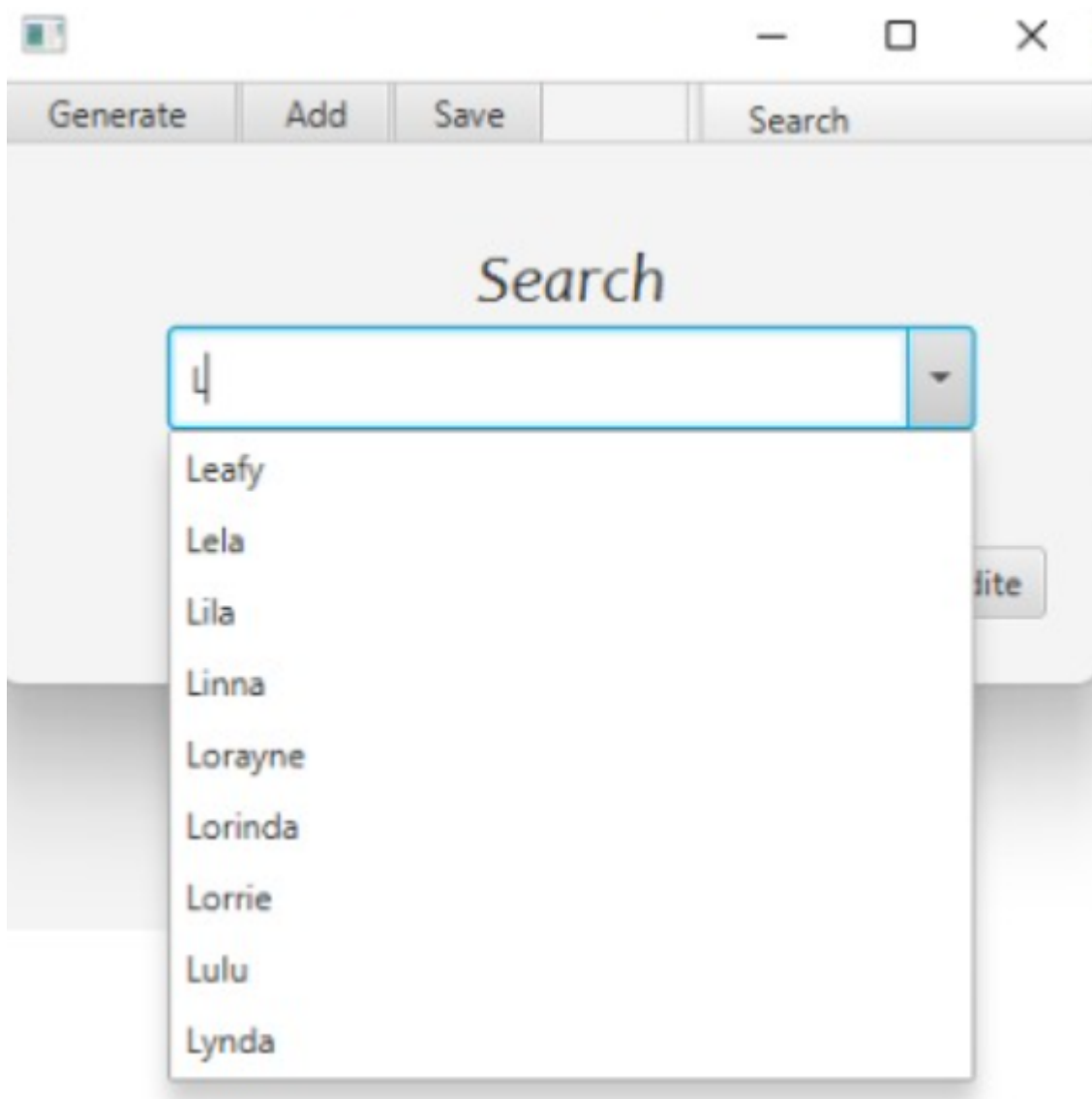
A screenshot of a software window titled "Add a Person". The window has a standard Windows-style title bar with minimize, maximize, and close buttons. Below the title bar is a menu bar with four items: "Generate", "Add", "Save", and "Search". The main content area of the window is titled "Add a Person" in a large, italicized font. Below the title, there are several input fields and a button. The first row contains two text input fields labeled "Name" and "L. Name". The second row contains a date input field labeled "Birthdate" with a calendar icon to its right, and a dropdown menu labeled "Gender". The third row contains a dropdown menu labeled "Nationality". The fourth row contains the label "Height:" followed by a text input field labeled "Meters". At the bottom right of the form is a button labeled "Create".

Interfaz de búsqueda (Igual para todos los criterios):



A screenshot of a software window titled "Search". The window has a standard Windows-style title bar with minimize, maximize, and close buttons. Below the title bar is a menu bar with four items: "Generate", "Add", "Save", and "Search". The main content area of the window is titled "Search" in a large, italicized font. Below the title is a single search input field with a dropdown arrow on the right side. The input field contains three dashes "---".

Lista emergente:



Interfaz de edición de registros:

Generate

Add

Save

Search

Edit or Delete data

Leafy

DIMARANAI

Birthdate

2002-12-11

Gender

FEMALE

Nationality

HAITI

Heigth:

1.5

Code:

57d7a916-...

Delete

Edit

Casos de prueba:

Objetivo de la Prueba: Se comprobarán las funcionalidades de un árbol AVL				
Clase	Método	Escenario	Valores de Entrada	Resultado
ArbolBinario	insert	escenario1 escenario2 escenario3	1	El árbol esta balanceado, se consiguen los elementos solicitados y el tamaño es el indicado

ArbolBinario	delete	escenario1 escenario2	1 2 230	Se eliminarán los elementos solicitados y el tamaño del árbol es uno menos que antes de eliminar y no elimina un valor que no se encuentra en el árbol además el árbol permanecerá balanceado
ArbolBinario	search	escenario1 escenario2	1 2 230	Se encuentran los elementos necesarios y no se encuentran los que no están en el árbol

Objetivo de la Prueba: Se comprobarán las funcionalidades de un árbol AV				
Clase	Método	Escenario	Valores de Entrada	Resultado
ArbolBinarioAV	get	escenario1 escenario3	0 10	El árbol no devuelve objetos que no existan y retornara el elemento solicitado

ArbolBinarioAV	delete	scenario1 scenario2	1 2 56	Se eliminarán los elementos solicitados y el tamaño del árbol es uno menos que antes de eliminar y no elimina un valor que no se encuentra en el árbol
ArbolBinarioAV	search	scenario1 scenario2	1 1000 100	Se encuentran los elementos necesarios y no se encuentran los que no están en el árbol

ArbolBinarioAV	size	scenario1 scenario2 scenario3		Los tamaños de cada árbol son correctos
ArbolBinarioAV	height	scenario1 scenario3		La altura de un árbol vacío es 0 y la altura del otro árbol es 4

Objetivo de la Prueba: Se comprobarán las generación de personas y que estas se unan a los cuatro árboles.

Clase	Método	Escenario	Valores de Entrada	Resultado
-------	--------	-----------	--------------------	-----------

DataBase	generatePeople	scenario1	1000	Se añaden a los cuatro árboles las personas generadas
----------	----------------	-----------	------	---

Diagrama de clases

Archivo en PDF:

[DataBase CRUD - Class Diagram](#)

Archivo en VPP:

[DataBase CRUD - VPP Class Diagram](#)

Archivo como imagen JPG:

