Mejorando el Encapsulamiento y Pruebas Unitarias en el Sistema de Gestión de Donantes de Sangre de la Cruz Roja



Valentina Montezuma González (2242058)

- (-)

- (-)

- (-)

Universidad del Valle

Mejorando el Encapsulamiento y Pruebas Unitarias en el Sistema de Gestión de Donantes de Sangre de la Cruz Roja

Valentina Montezuma González (2242058)

- (-)

- (-)

- (-)

Profesor

Víctor Andrés Bucheli

Taller Programación Orientada a Objetos

Universidad del Valle

Facultad de Ingeniería

Programación Orientada a Objetos

Santiago de Cali

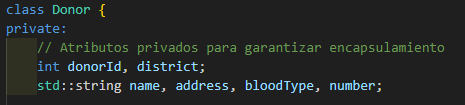
2025

Taller Encapsulamiento y Pruebas Unitarias

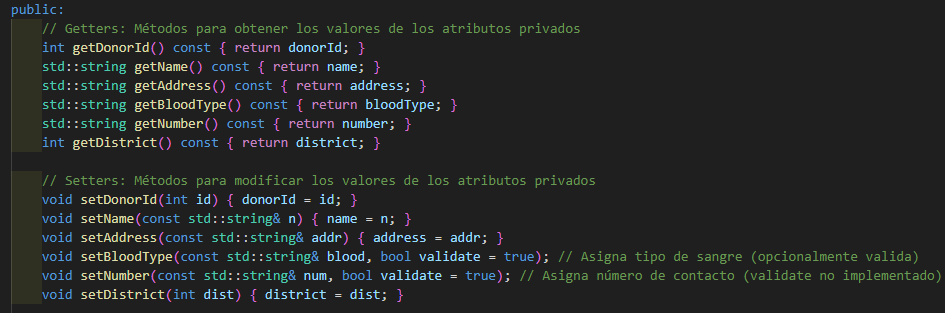
Análisis del diseño de clases, código modificado y problemas resueltos.

Como problema inicial que identificamos fue que los atributos dirección, donorId, name, address, distric, bloodType y number de la clase Donor eran públicos aun tratándose de datos de información delicada, lo que no cumplía con el principio de encapsulamiento, por lo que realizamos los siguientes cambios:

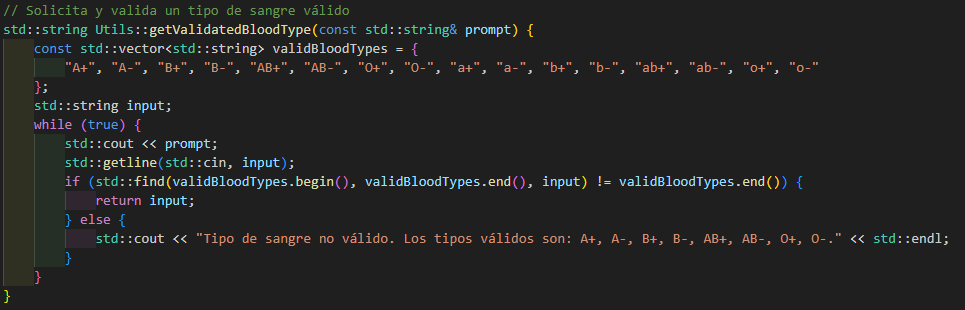
1. Pusimos los atributos privados.

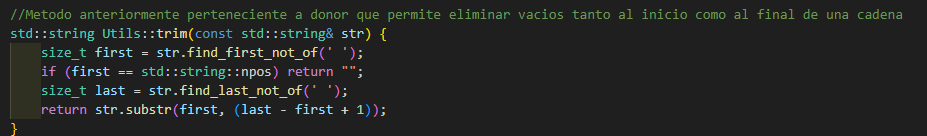


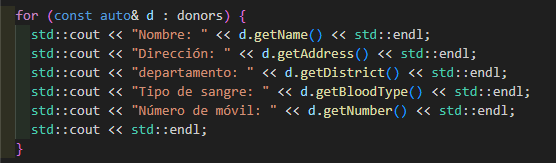
1. Se agregaron setters y getters para acceder a los atributos y modificarlos de una manera controlada y sin acceder directamente a ellos.



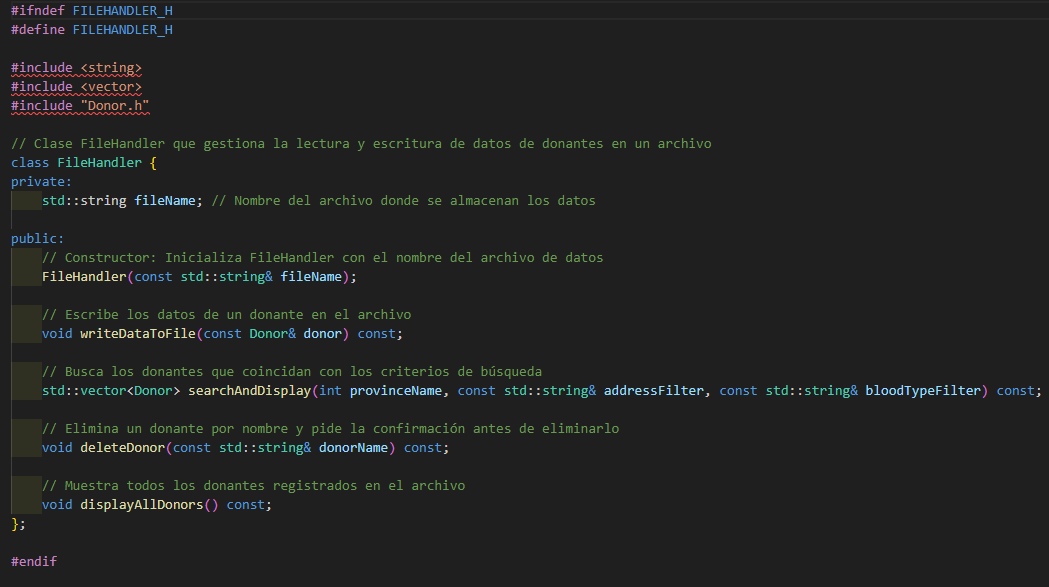
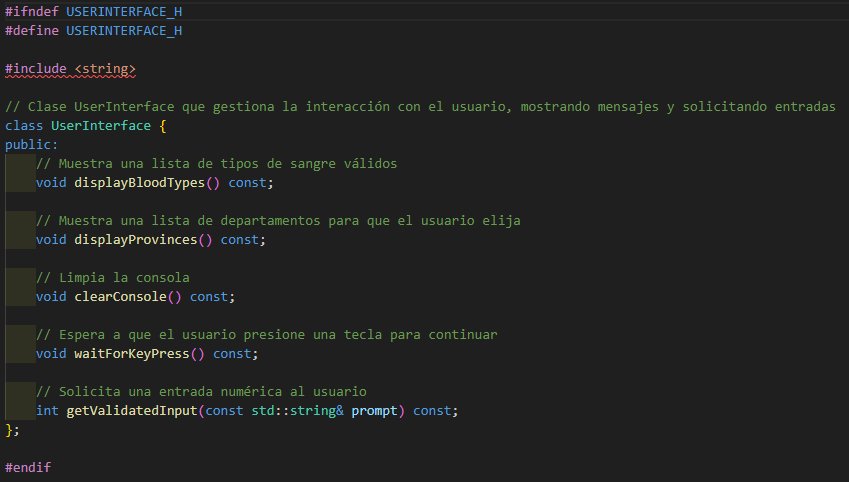
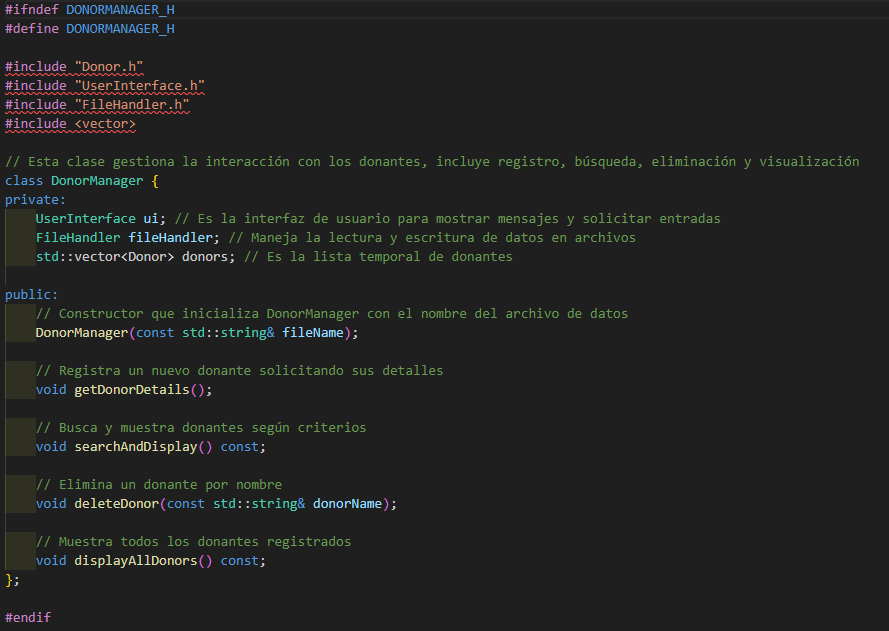
1. Implementamos validaciones para asegurar que los datos sean correctos (El número de teléfono mediante la API y el tipo de sangre usando los tipos existentes). Esto ya que anteriormente había errores al tomar el celular y en el caso de la sangre se podía poner cualquier letra haciendo que el programa no fuera preciso. Cabe aclarar que esta validación se encuentra en una nueva clase llamada Utils.



1. Movimos el método trim de la clase Donor ya que podía usarse más allá de la misma, por lo que se puso en la nueva clase de tils, la que hablaremos a profundidad mas adelante.

Algunas clases accedían a los datos privados de Donor, así que se asignó que en su lugar accediera a los getters para evitar errores y respetar el encapsulamiento, a continuación, un ejemplo en el que se accede a los atributos mediante sus gets. (Fue un ejemplo ya que esto ocurre a lo largo de todo el código). 

Por otra parte, la clase BloodDatabase era extremadamente grande y manejaba diferentes responsabilidades que terminaba por dificultar su comprensión y adecuación, por lo que decidimos modular la misma para mejorar la estructura del código. Se crearon tres nuevas clases a partir de ella donde se dividieron sus funciones separando las responsabilidades, a continuación, una muestra de ellas:

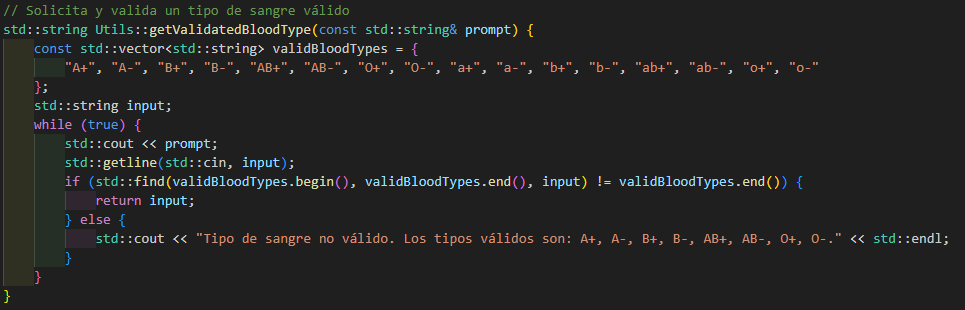
* Clase FileHanlder: Es la encargada de manejar la lectura y la escritura de los datos en los archivos, asegurando que haya una persitencia en la información de los donantes. Esta escribe nuevos registros, busca donantes según criterios, elimina entradas y muestra el historial completo. Lo que hace es centralizar todas las operaciones relacionadas con archivos para mantener los datos organizados.
* Clase UserInterface: Es la que se encarga de gestionar lo relacionado a la interacción con el usuario, mostrando mensajes, menus y solicitando entradas validadas. Entre sus funcionalidades esta limpiar la consola, esperar la interacción del usuario y hacer las validaciones mediante UItils. Lo que hace es separar la lógica de presentación y entrada de datos, proporcionando una experiencia de usuario clara y entendible.
* Clase DonorManager: Es la que se encarga de coordinar la lógica relacionada con los donantes, utilizando FileHandler para manipular datos y UserInterface para interactuar con el usuario. Sus funcionalidades incluyen registrar nuevos donantes, buscar y mostrar registros, y eliminar donantes. Su objetivo es actuar como el núcleo que conecta las operaciones de archivos y la interfaz de usuario.

También se agregaron los métodos de:

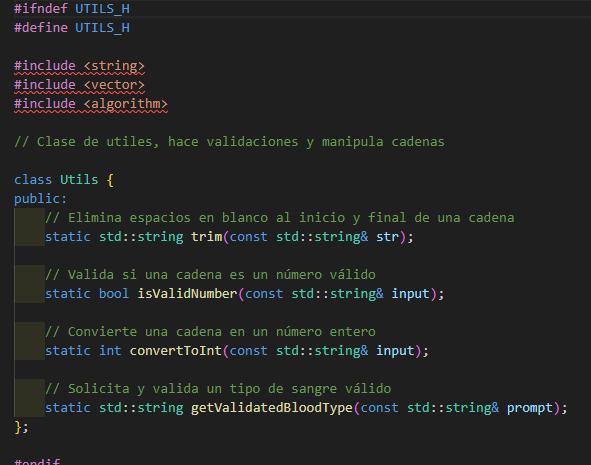
1. Porcentaje de cada tipo de sangre en la base de datos
2. Tipo de sangre más común en la base de datos
3. Porcentaje de la población de cada tipo de sangre
4. Que tipo de sangre tiene la mayoría de la gente

Descripción de la reutilización de código.

El método trim, como se mencionó anteriormente, se movió a una clase totalmente nueva, de manera que se facilita su reutilización en cualquier parte del proyecto y deja de estar limitado para una sola clase.

De igual manera se movió la clase getValidatedInput anteriormente de BloodDatabase a Utils, esto para darle un uso más general y a su vez se utilizó parte de su código para crear la validación de los tipos de sangre, pues anteriormente no se verificaba si el dato ingresado por el usuario era un tipo de sangre valido, a continuación, se muestra como quedo estructurado en la clase Utils: 

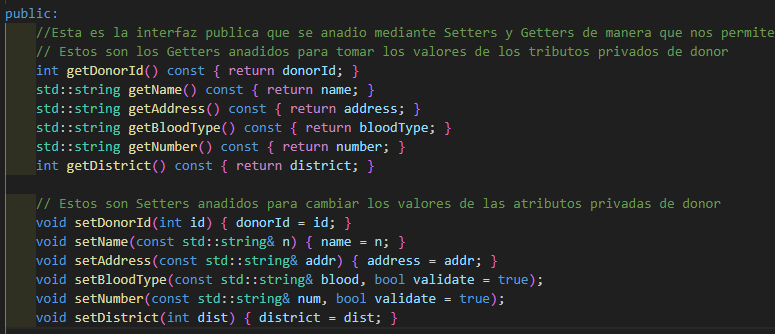
Se creo una nueva clase Utils la cual es una clase de utilidad que proporciona métodos que permiten validar y manipular las cadenas, como eliminar espacios en blanco, validar números y tipos de sangre y convertir cadenas en números. Al requerir diferentes tipos de validaciones, las nuevas se agregaron aquí, y como se hablo previamente en el trabajo, las validaciones que estaban anteriormente en otras clases se movieron a esta para respetar el principio de abstracción. A continuación se muestra como quedo definida esta nueva clase:



Descripción de la API integrada.

Implementación de interfaces publicas

Los getters y setters se crearon con el propósito de actuar como interfaz publica, pues son los que se usan a lo largo de BloodDatabase para acceder a los a tributos privados de Donor. Es decir, permiten su reutilización y a la vez mantener el encapsulamiento de manera correcta.



Código de las pruebas unitarias y resultados de las mismas