

```
class Vehiculo(abc):
```

```
    def moverse(self):
```

```
    | Vehiculo |
```

```
    | + moverse() |
```

```
    | + detenerse() |
```

```
    | Coche | |Bicicleta|
```

```
    | + moverse() |
```

```
    | + detenerse() |
```

```
    | + moverse() |
```

```
    | + detenerse() |
```

Caso de Estudio: Sistema de Gestión de Donantes de Sangre: Cruz Roja



Caso de Estudio: Sistema de Gestión de Donantes de Sangre: Cruz Roja

El siguiente documento ha sido elaborado tomando fragmentos de documentos previamente publicados, así como imágenes y ejercicios, sobre los cuales se han realizado traducciones libres y adaptaciones con el propósito de desarrollar el material para el curso de Fundamentos de Programación Orientada a Objetos de la Universidad del Valle. Se ha respetado la autoría original de los materiales, y se han proporcionado las referencias correspondientes dentro del documento. El uso de este material se restringe estrictamente al marco académico y formativo de los estudiantes que cursen la asignatura, y no tiene fines comerciales. Queda prohibida su copia o distribución fuera del contexto de dicho curso.

Caso de Estudio: Sistema de Gestión de Donantes de Sangre: Cruz Roja

Introducción

El paradigma de la programación orientada a objetos (POO) se erige como uno de los enfoques más populares y efectivos en el ámbito del desarrollo de software [1]. Al modelar el mundo real en objetos que interactúan entre sí, la POO nos permite crear sistemas más intuitivos, flexibles y mantenibles [2]. En este estudio de caso, centrado en un sistema de gestión de donantes de sangre de la Cruz Roja, veremos cómo la POO, a través de conceptos como la herencia, el polimorfismo y la encapsulación, facilita la creación de software de alta calidad y adaptable a los cambios [3]. Este es un proyecto del curso FPOO de la Universidad del Valle, se desarrolla con fines académicos, puede ser utilizado con fines académicos dando los créditos de la autoría. El código se basó en el proyecto BloodDatabase [4] se tradujo al español y se adaptó a las necesidades del curso.

Contexto del caso

Imaginemos que somos contratados para **desarrollar un sistema de gestión de donantes de sangre para la Cruz Roja Colombiana**. Este sistema es capaz de registrar nuevos donantes, buscar donantes existentes y eliminar donantes del sistema. Incluye menús que facilitan la navegación, almacena la información de manera estructurada en archivos planos y posee validaciones básicas que garantizan el correcto ingreso de los datos. El código, compuesto por aproximadamente 350 líneas, se organiza en dos clases y un archivo principal: Donantes, Banco de Sangre y Main. El código fue entregado a **Colombia** por la **Cruz Roja Internacional**, por lo que nuestra tarea consiste en desarrollar tres nuevos requerimientos del sistema y resolver un BUG (error del software entregado).

Descripción del Banco de Sangre

El proyecto titulado: **Sistema de Gestión de Donantes de Sangre** está diseñado para facilitar tanto la gestión como el almacenamiento de la información de los donantes de sangre. Esta aplicación ofrece una interfaz de usuario intuitiva que permite registrar, buscar y eliminar donantes de manera eficiente. El sistema resulta ideal para organizaciones de

Caso de Estudio: Sistema de Gestión de Donantes de Sangre: Cruz Roja

salud y bancos de sangre que requieren mantener un registro organizado y accesible de sus donantes. En la Figura 1. se presentan las pantallas correspondientes a cada menú, y a continuación se detallan las tareas básicas que se pueden realizar con el sistema:

- Pantalla de inicio: se presenta el menú principal.
- Registro de Donante: la opción 1 y se ingresan los detalles del nuevo donante. Estos detalles se guardan en el archivo data.txt.
- Búsqueda de Donantes: la opción 2 y se ingresan los criterios de búsqueda. El programa muestra los donantes que coinciden con los criterios ingresados.
- Eliminación de Donante: la opción 3, se ingresa el nombre del donante a eliminar, y se confirma la eliminación.
- Salida del Programa: la opción 4 para salir del programa.


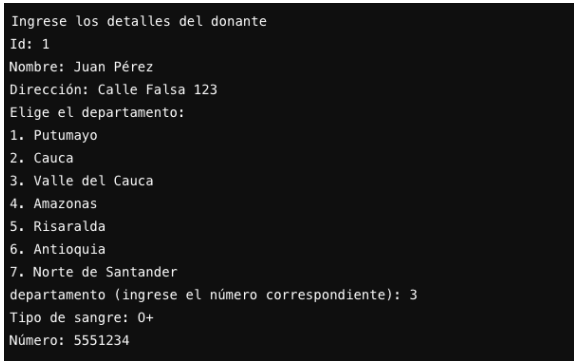


Menú principal	Ingresar Donante
 <pre> 1. Registrar donante 2. Buscar donante 3. Eliminar donante 4. Salir Ingrese su elección: 1 </pre>	 <pre> Ingrese los detalles del donante Id: 1 Nombre: Juan Pérez Dirección: Calle Falsa 123 Elige el departamento: 1. Putumayo 2. Cauca 3. Valle del Cauca 4. Amazonas 5. Risaralda 6. Antioquia 7. Norte de Santander departamento (ingrese el número correspondiente): 3 Tipo de sangre: 0+ Número: 5551234 </pre>
Buscar Donante	Eliminar donante
 <pre> 1. Registrar donante 2. Buscar donante 3. Eliminar donante 4. Salir Ingrese su elección: 2 Elige el departamento: 1. Putumayo 2. Cauca 3. Valle del Cauca 4. Amazonas 5. Risaralda 6. Antioquia 7. Norte de Santander Ingrese el número de la departamento: 3 Ingrese la dirección (dejar en blanco para omitir): Ingrese el tipo de sangre (dejar en blanco para omitir): Personas de la departamento 3: Nombre: Juan Pérez Dirección: Calle Falsa 123 departamento: 3 </pre>	 <pre> 1. Registrar donante 2. Buscar donante 3. Eliminar donante 4. Salir Ingrese su elección: 3 Ingrese el nombre del donante a eliminar: Juan Pérez Nombre: Juan Pérez Dirección: Calle Falsa 123 Tipo de sangre: 0+ Número de móvil: 5551234 ¿Está seguro de que desea eliminar al donante? [s/n]: s </pre>

Figura 1. Pantallas de cada menú del banco de sangre

Caso de Estudio: Sistema de Gestión de Donantes de Sangre: Cruz Roja

Detalles de la implementación

La implementación del **Sistema de Gestión de Donantes de Sangre** se basa en dos clases principales: **Donor** y **BloodDatabase**. La clase **Donor** representa a un donante individual, con atributos como **donorId**, **name**, **address**, **district**, **bloodType** y **number**. Esta clase incluye métodos para mostrar los detalles del donante.

Por otro lado, la clase **BloodDatabase** gestiona la colección de donantes y proporciona métodos para agregar nuevos donantes (**getDonorDetails** y **writeDataToFile**), buscar y mostrar donantes (**searchAndDisplay**), y eliminar donantes (**deleteDonor**). Además, incorpora funciones auxiliares como **clearConsole**, **waitForKeyPress** y **getValidatedInput**, que mejoran la interacción con el usuario.

El programa principal (main) ofrece un menú interactivo que permite a los usuarios registrar, buscar y eliminar donantes, ejemplificando la organización y eficiencia del código orientado a objetos. El diagrama de clases se presenta en la Figura 2, y el código fuente está disponible las siguientes direcciones:

1. <https://replit.com/@vbucheli/CasoEstudioBancoSangre>
2. <https://github.com/vbucheli/CasoEstudioBloodDatabase>

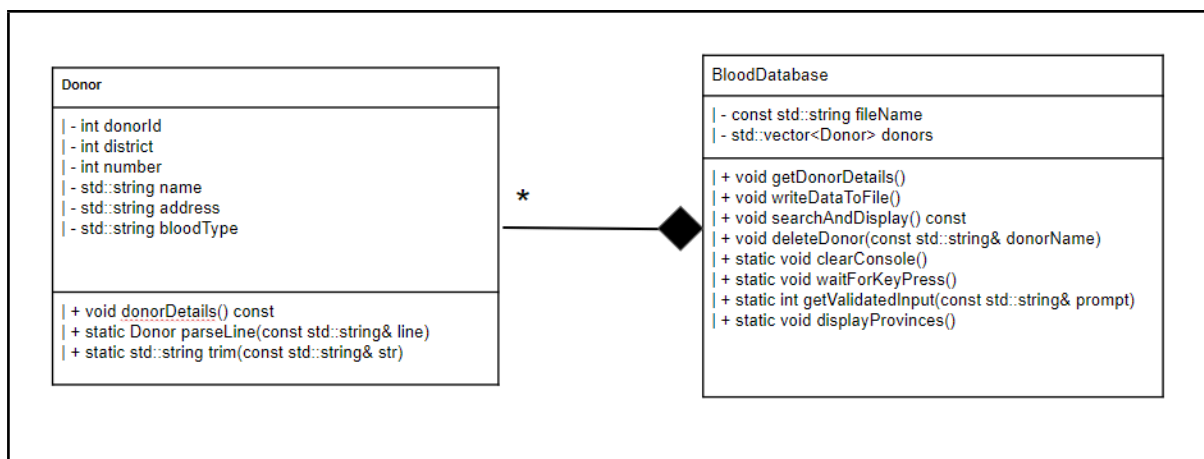


Figura 2. Diagrama de clases del banco de sangre

Caso de estudio

Para este proyecto, ustedes son contratados como programadores para desarrollar el sistema de gestión de donantes de sangre **Cruz Roja Colombia** con los siguientes requerimientos:

1. Al ingresar un nuevo donante, además de los departamentos ya existentes, el usuario deberá tener la opción de seleccionar entre los siguientes departamentos: Chocó, Arauca y Guainía, en el menú "Elige el departamento".



Caso de Estudio: Sistema de Gestión de Donantes de Sangre: Cruz Roja

2. Presentar al usuario los tipos de sangre como una lista: A+, A-, B+, B-, AB+, AB-, O+ y O-.
3. Resolver los siguiente BUG:
 - i. Si al ingresar un número de celular válido, el sistema muestra el siguiente error: "Entrada fuera de rango. Por favor, ingrese un número válido."
Un ejemplo es utilizar el número celular válido 3002176362 el sistema arroja el error.
Esto indica que el sistema está generando un error para los números válidos.
4. Como usuario, es necesario que exista una opción en el menú que permita ver el historial de donaciones [punto avanzado].

Preguntas de análisis

Evaluación de las ventajas de la codificación en el Paradigma Orientado a Objetos:

1. Modularidad: ¿Cuáles módulos o clases identifica en el caso? ¿Considera que cada clase representa un objeto más pequeño y manejable? ¿En cuál sección del código encuentra que el programa está dividido?
2. Evalúe si la siguiente expresión es verdadera o falsa y justifique su respuesta:
La modularidad facilita la localización y corrección de errores. Si hay un problema en una clase, se puede abordar sin afectar al resto del sistema.
3. Evalúe si la siguiente expresión es verdadera o falsa y justifique su respuesta:
Los desarrolladores pueden trabajar en diferentes clases simultáneamente sin conflictos, mejorando la productividad y la eficiencia del equipo de desarrollo.
4. Reusabilidad: ¿Qué código ha reutilizado hasta el momento?
5. Evalúe si la siguiente expresión es verdadera o falsa y justifica tu respuesta:
Las clases pueden ser reutilizadas en diferentes partes del programa o incluso en otros proyectos, reduciendo el esfuerzo de desarrollo.
6. Evalúe si la siguiente expresión es verdadera o falsa y justifique su respuesta:
El ocultamiento de la información permitió resolver los requerimientos del caso.
7. ¿Considera que en el caso hay **consistencia**, pues la reutilización de código asegura que las funcionalidades compartidas se comporten de manera uniforme en todo el sistema?
8. Extensibilidad: ¿Qué nuevas funcionalidades adiciona a la aplicación? ¿considera que es fácil añadir nuevas funcionalidades y extender las existentes?
9. ¿Considera que el código del caso es claro y organizado? ¿La Mantenibilidad



Caso de Estudio: Sistema de Gestión de Donantes de Sangre: Cruz Roja

facilita la comprensión y modificación del sistema?

10. ¿Considera que el código del caso es sencillo de actualizar y mejorar? ¿Esto se logra por la separación de responsabilidades entre las clases?
11. Facilidad de comprensión ¿El código entregado en el caso está escrito de la forma en que nosotros entendemos y modelamos el mundo real? ¿Es intuitivo el uso de objetos y relaciones entre ellos?
12. Se puede argumentar que:

El software proporciona resultados correctos o precisos.

Referencias

- [1] M. A. Weisfeld, *The object-oriented thought process*, 3rd ed. en Developer's library. Upper Saddle River, NJ: Addison-Wesley, 2009.
- [2] J. A. Villalobos S., *Fundamentos de programación: aprendizaje activo basado en casos : un enfoque moderno usando Java, UML, objetos y eclipse*. Colombia: Pearson, 2006.
- [3] Y. Bugayenko, *Elegant objects. volume 1*, Version: 1.7. Palo Alto, California: Yegor Bugayenko, 9.
- [4] S. Khorshed, *BllodDatabase*. [En línea]. Disponible en: <https://github.com/khorshedsadhin/Blood-Bank-Management-System-Project>