


```
class Vehiculo {  
    + moverse()  
    + detenerse()  
};
```

```
| Vehiculo |  
| + moverse() |  
| + detenerse() |
```

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

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

Semántica orientada a objetos en C++



Semántica orientada a objetos en C++

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.

Semántica orientada a objetos en C++

A partir del estudio de caso sobre el Sistema de Gestión de Donantes de Sangre de la Cruz Roja, se solicita crear un módulo de reportes, estadísticas e indicadores. El problema es adaptado y modificado de [1] y se solicita que el dicho módulo sea accesible desde el menú principal. En el sistema se conoce la siguiente información de las personas: el tipo de sangre, procedencia y edad.

El tipo de sangre y la procedencia se manejan de acuerdo a las siguientes convenciones:


Tipo de sangre: A+, A-, B+, B-, AB+, AB-, O+, O-

Procedencia:

1. Putumayo
2. Cauca
3. Valle del Cauca
4. Amazonas
5. Risaralda
6. Antioquia
7. Norte de Santander
8. Chocó
9. Arauca
10. Guainía

El reporte debe contener:

1. Determinar el total de personas en el sistema.
2. Determinar el número de chocoanos con tipo de sangre A+, araucanos con tipo de sangre O- y caucanos con tipo de sangre A+.
3. Determinar la edad promedio de los individuos de Choco, Arauca y Valle del Cauca.



Semántica orientada a objetos en C++

4. Determinar la cantidad de vallecaucanos con tipo de sangre B+ y mayores de edad.
5. Para el caso debe tomar como base el siguiente proyecto, disponible en los siguientes enlaces:
 - a. <https://replit.com/@vbucheli/CasoEstudioBancoSangre>
 - b. <https://github.com/vbucheli/CasoEstudioBloodDatabase>

Acompañamiento para la solución

Abstracción

La abstracción es el proceso de simplificar la complejidad de un sistema al centrarse en los aspectos esenciales y relevantes de los objetos, mientras se ocultan los detalles innecesarios [2]. En el Sistema de Gestión de Donantes de Sangre de la Cruz Roja, la abstracción se refleja en la identificación de funcionalidades clave, como el cálculo del total de donantes o la generación de estadísticas sobre las edades de los donantes. Estos métodos representan las acciones esenciales que un usuario necesita realizar, sin necesidad de conocer cómo se gestionan internamente los datos o cómo se procesan las listas de donantes. De esta forma, la abstracción permite que el usuario interactúe con el sistema de manera eficiente, enfocándose en los resultados importantes y dejando los detalles técnicos a la implementación interna.

De igual manera, la abstracción facilita el proceso de convertir los requerimientos en clases, métodos o interacciones de éstos, al identificar las funcionalidades clave que deben ser implementadas en el sistema, eliminando los detalles irrelevantes. En el caso del Sistema de Gestión de Donantes de Sangre de la Cruz Roja, cada requerimiento, como calcular el número de donantes por tipo de sangre o determinar la edad promedio por región, se abstrae en métodos que integran la lógica necesaria para cumplir con esas tareas. Estas funcionalidades se agrupan en clases que representan conceptos del mundo real, como Donor o Reports (tal como veremos más adelante), lo que permite organizar el código de manera clara y eficiente. En este caso se podría construir la clase Reports y un método `donorTotal()` el cual abstrae el cálculo del total de donantes, ocultando los detalles de la iteración sobre los datos y la validación de condiciones específicas, facilitando así la conexión entre los requerimientos y su implementación en el código.

A continuación se describen las fases para resolver un problema en el paradigma orientado a objetos, el proceso planteado en [1] se modifica y adapta al problema específico del caso Sistema de Gestión de Donantes de Sangre de la Cruz Roja, las fases son:

- a) Construcción de la tabla de requerimientos.
- b) Abstracción de clases o ***diagramas de clases***
- c) Descripción de las responsabilidades de las clases, formalizadas en los contratos de cada método.
- d) Escritura de código orientado a objetos.

Semántica orientada a objetos en C++

Según [1] y [2] la creación de la tabla de requerimientos es un paso fundamental en el análisis del problema. Los requerimientos representan las necesidades del usuario, es decir, señalan los aspectos que se buscan resolver a través de la aplicación. Acá se identifican y describen aquellos aspectos que están vinculados directamente con el comportamiento o funcionalidad del sistema, a estos se les conoce como requerimientos funcionales. Según [1] la tabla de requerimientos está compuesta por cuatro columnas:

1. Identificación del requerimiento: es un código que identifica al requerimiento
2. Descripción: consiste en una descripción concisa y clara, en lenguaje natural, del requerimiento.
3. Entradas: son los insumos o datos necesarios para que el requerimiento se pueda suplir con éxito.
4. Resultados o salidas: constituyen el cumplimiento del requerimiento, es decir, son los resultados que dan solución a un requerimiento funcional definido por el usuario.

Identificación del requerimiento	Descripción	Entradas	Resultados (Salidas)
R1	Conocer la cantidad de personas a procesar	Los datos de las donantes en el sistema	La cantidad de personas a procesar almacenada en una variable (n)
R2	Conocer los tipos de sangre, procedencias y edades de todas las personas	Un número entero calculado a partir de los datos del sistema de las personas: su tipo de sangre, procedencia y edad	La cantidad de personas a procesar almacenada de acuerdo con su tipo de sangre, procedencia y edad
R3	Encontrar el número de chocoanos con tipo de sangre A+	Los datos de todas las personas	El número de chocoanos con tipo de sangre A+
R4	Encontrar el número de araucanos con tipo de sangre O-	Los datos de todas las personas	El número de araucanos con tipo de sangre O-
R5	Encontrar el número de vallecaucanos con tipo de sangre B+	Los datos de todas las personas	El número de caucanos con tipo de sangre B+
R6	Hallar la edad promedio de los individuos de otras procedencias (Choco, Arauca y Valle del Cauca)	Los datos de todas las personas	La edad promedio de los individuos de (Choco, Arauca y Valle del Cauca)

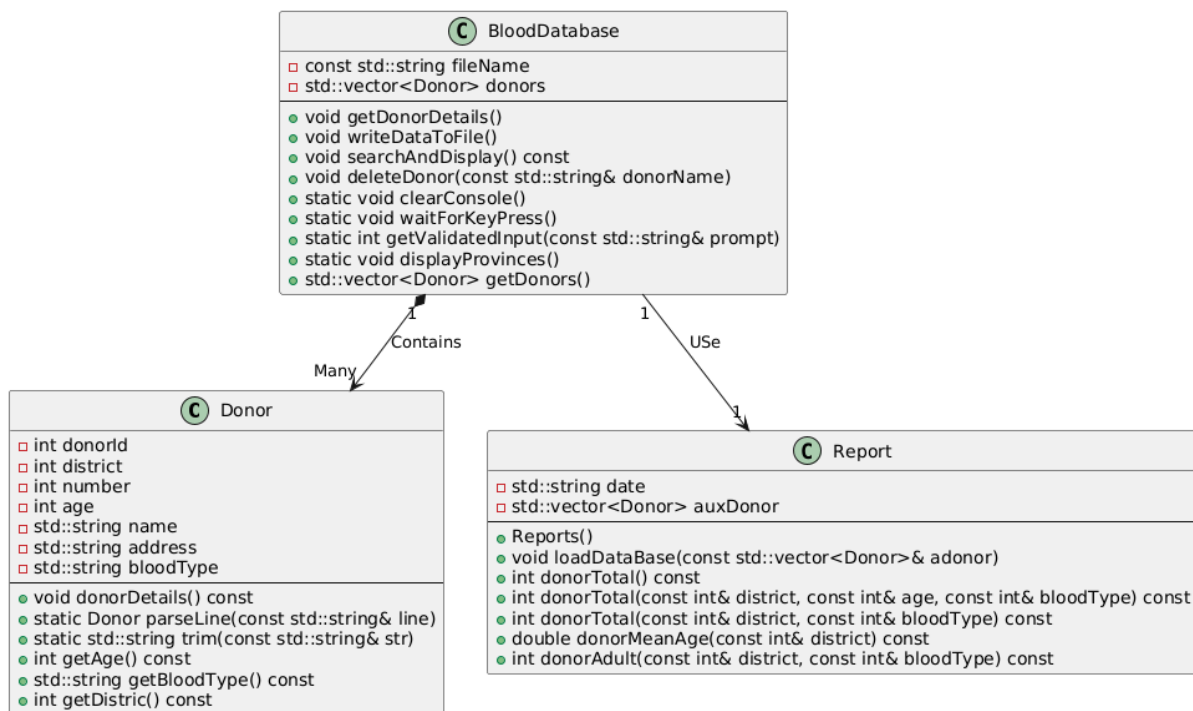
Semántica orientada a objetos en C++

R7	Hallar la cantidad de vallecaucanos con tipo de sangre B+ y mayores de edad.	Los datos de todas las personas	La cantidad de vallecaucanos con tipo de sangre B+ y mayores de edad.
----	--	---------------------------------	---

Table 1: Tabla de requerimientos del caso Reportes-Sistema de Gestión de Donantes de Sangre de la Cruz Roja.

Abstracción de clases o diagrama de clases

La abstracción de clases es una parte fundamental del análisis del problema y representa el primer paso hacia el diseño de la solución [3]. Se trata de una representación gráfica del problema, un esquema de software, en el que se modelan abstracciones de la realidad que están relacionadas con el dominio del problema y que pueden ser implementadas mediante software [1].



Recuadro 1: Diagrama de clases - Reportes-Sistema de Gestión de Donantes de Sangre de la Cruz Roja

El diagrama de clases para el sistema de gestión de donantes de sangre de la Cruz Roja representa la estructura y comportamiento de las entidades involucradas en el sistema, en negrita los cambios realizados al diagrama de clases del caso anterior, la clase *Report* esta todo en negrita dado que es completamente nueva. A continuación la descripción del diagrama de clases, la clase principal, **'Donor'**, contiene los datos de los donantes, como el nombre, tipo de sangre, procedencia, edad, y otros detalles personales. Estos atributos cuentan con métodos de acceso. Por ejemplo, los métodos como **'getAge()'** y **'getBloodType()'** permiten acceder a la información sin exponer directamente los atributos, asegurando que los datos sensibles estén correctamente gestionados.

Además, el diagrama incluye la clase **'BloodDatabase'**, que actúa como un administrador

Semántica orientada a objetos en C++

de los donantes, encargándose de funciones como la carga de datos, escritura en archivo y búsqueda de información. Esta clase interactúa con los objetos de la clase `Donor`, aplicando métodos para gestionar la base de datos de donantes. También define métodos estáticos como `clearConsole()` y `waitForKeyPress()`, que mejoran la interacción del usuario con el sistema. Estos métodos son invocados cuando el usuario necesita realizar tareas como agregar, eliminar o buscar donantes, lo que convierte a `BloodDatabase` en una clase fundamental para el funcionamiento del sistema.

Por último, la clase `Reports` genera informes estadísticos sobre los donantes en función de criterios específicos, como el tipo de sangre, procedencia o edad. A través de métodos como `donorTotal()` y `donorMeanAge()`, se puede calcular el número de donantes en diferentes categorías y promediar sus edades, entre otras estadísticas. Esta clase toma un vector de objetos `Donor` desde `BloodDatabase` y los procesa para generar los reportes. La interacción entre estas clases sigue el diseño orientado a objetos, donde cada una tiene una responsabilidad específica y se comunican a través de métodos bien definidos, permitiendo un sistema modular y fácilmente mantenible.

El análisis de responsabilidades de las clases

De acuerdo con [2] esta etapa implica describir los métodos de cada clase a través de contratos que detallan los requerimientos correspondientes, la precondition que indica el estado del objeto antes de ejecutar el método, la postcondición que especifica el estado del objeto tras la ejecución, y un modelo verbal que ofrece una descripción en lenguaje natural de la solución propuesta, similar a lo que se conoce como un algoritmo cualitativo. La asignación de responsabilidades es parte fundamental de la documentación de la solución o del sistema de software que se desarrollará.

Análisis de responsabilidades de clases: Contrato de la clase reporte

Nombre del método	Requerimiento o asociado	Precondición	Postcondición	Modelo verbal
loadDataBase	R1	Se desconocen los datos de las personas	Se conocen todos los datos	- Tomar el vector de donantes desde la clase database y lo carga en auxdonante de la clase reporte

Semántica orientada a objetos en C++

donorTotal	R1,R 2,R3, R4,R 5	Se desconocen los datos estadísticos de las personas	Se conocen todos los datos estadísticos de las personas por diferentes opciones distrito, edad y tipo de sangre	<ul style="list-style-type: none"> - Se recorre el vector de donante - El contador se inicia en 0 - Si cumple con las características un contador se incrementa - Se retorna el contador
donorMeanAge	R6	Se desconocen los datos de edad de las personas	Se conocen todos los datos de las edades de las personas por diferentes opciones distrito, edad y tipo de sangre	<ul style="list-style-type: none"> -Se recorre el vector de donante -El contador se inicia en 0 -Se inicia 0 una sumatoria de edades -Se inicia 0 si cumple con las características -Un contador se incrementa -la edad se suma como un acumulador -Se retorna el (sumatoria de edad dividido contador
donorAdult	R7	Se desconocen los datos de edad de las personas	Se conocen todos los datos de las edades de las personas por diferentes opciones distrito, edad y tipo de sangre	<ul style="list-style-type: none"> -Se recorre el vector de donante -El contador se inicia en 0 -Si cumple con las características y la edad es mayor igual a 18 un contador se incrementa -Se retorna el contador

Tabla 2: Codificación- Reportes-Sistema de Gestión de Donantes de Sangre de la Cruz Roja

Encapsulamiento

El encapsulamiento es uno de los principios fundamentales de la programación orientada a objetos, y se refiere a la capacidad de ocultar los detalles internos de una clase y proteger los datos que maneja [4]. En C++, esto se logra definiendo los atributos como `private` o `protected`, y permitiendo el acceso a estos únicamente a través de métodos públicos o funciones miembro, lo que asegura un control estricto sobre cómo se manipulan los datos [5].

En el contexto del Sistema de Gestión de Donantes de Sangre de la Cruz Roja la clase Report aplica el encapsulamiento dado que los atributos como `date`, o la información de los donantes(`vector<Donor> auxDonor`) se declaran como privados. Esto garantiza que estos datos sólo puedan ser modificados o consultados a través de métodos públicos controlados, como `loadDataBase()`. El encapsulamiento se aplica para proteger la manera en que los datos de los donantes son procesados y presentados en los reportes estadísticos. La clase

Semántica orientada a objetos en C++

Report contiene métodos que realizan cálculos sobre los donantes, como el total de donantes, la edad promedio y la cantidad de donantes de un tipo de sangre específico, y se asegura de que la lógica interna de estos cálculos esté encapsulada dentro de la clase, para que no sea accesible desde fuera.

De otra parte el encapsulamiento se refiere a la ocultación de los detalles internos de un objeto. En lugar de exponer cómo funcionan los métodos o cómo se gestionan los datos internamente, el encapsulamiento garantiza que los usuarios de una clase interactúan sólo con las interfaces públicas [6]. Esto permite proteger los datos sensibles y mantener la integridad de los objetos. En C++, se logra mediante el uso de modificadores de acceso como *private* y *public*. Por ejemplo, en el Sistema de Gestión de Donantes de Sangre de la Cruz Roja, el módulo de estadísticas encapsula la lógica interna de cálculo de indicadores, permitiendo que los usuarios del sistema soliciten reportes sin preocuparse por cómo se realizan las operaciones, solo accediendo a métodos públicos como `donorTotal()` o `donorMeanAge()`.

En esta sección se presentan algunos aspectos de la implementación, el código está en los siguientes enlaces:

- <https://replit.com/join/hmsjyqiwc-fpoounivalle>
- <https://github.com/vbucheli/CasoEstudioReportBloodDatabase>.

Modificaciones en el menú

Se solicita crear un módulo de estadísticas e indicadores accesible desde el menú principal. En el archivo `main.cpp` verificar los siguientes cambios:

```
23         std::cout <<
24         "
25         "
26         "
27         "
28         "
29         "
30
31         std::cout << "1. Registrar donante\n";
32         std::cout << "2. Buscar donante\n";
33         std::cout << "3. Eliminar donante\n";
34         std::cout << "4. Reportes\n";
35         std::cout << "5. Salir\n";
36         std::cout << "Ingrese su elección: ";
37         std::cin >> choice;
38         std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); // descartar cualquier
39         entrada extra
```

Figura 1. Menú cruz roja

Semántica orientada a objetos en C++

```
40     switch (choice) {
41     case 1:
42         database.getDonorDetails();
43         database.writeDataToFile();
44         break;
45     case 2:
46         database.searchAndDisplay();
47         break;
48     case 3:
49         std::cout << "Ingrese el nombre del donante a eliminar: ";
50         std::getline(std::cin, donorName);
51         database.deleteDonor(donorName);
52         BloodDatabase::waitForKeyPress();
53         break;
54     case 4:
55         std::cout << "Reporteador " << endl << endl;
56         reporte.loadDataBase(database.getDonors());
57         std::cout << "El total de registros es: " << reporte.donorTotal() << endl << endl;
58         std::cout << "El total de registros del Valle del Cauca, 43 años y B+ es: " << reporte.donorTotal(d,a,b) << endl << endl;
59         d=8; std::cout << "El total de registros de Chocoanos B+ es: " << reporte.donorTotal(d,b) << endl << endl;
60         d=9; b=8; std::cout << "El total de registros de Aracaunos 0+ es: " << reporte.donorTotal(d,b) << endl << endl;
61         d=3; b=1; std::cout << "El total de registros de Valle Caucaños A+ es: " << reporte.donorTotal(d,b) << endl << endl;
62         double auxedad; d=8; auxedad=reporte.donorMeanAge(d);
63         d=9; auxedad=(auxedad+reporte.donorMeanAge(d))/2;
64         d=3; auxedad=(auxedad+reporte.donorMeanAge(d))/2;
65         std::cout << "El promedio de edad los individuos de Choco, Arauca y Valle de Cauca: " << auxedad << endl << endl;
66         d=3; b=3;
67         std::cout << "El total de registros mayores de edad vallecaucanos B+: " << reporte.donorAdult(d,b) << endl << endl;
68         break;
69     case 5:
70         std::cout << "Gracias por usar el Sistema de la Cruz Roja" << std::endl;
71         return 0;
72     default:
73         std::cout << "Opción no válida. Inténtalo de nuevo.\n";
74         BloodDatabase::waitForKeyPress();
75         break;
76     }
77 }
78 }
79 }
```

Recuadro 2. Switch de menú cruz roja

Implementación de las clases

La clase donante, base de datos y reporte

```
1  #ifndef DONOR_H
2  #define DONOR_H
3
4  #include <string>
5
6  class Donor {
7  public:
8      int donorId, district, bloodType, age;
9      std::string name, address, number;
10
11      void donorDetails() const;
12      int getAge() const;
13      int getBloodType() const;
14      int getDistrict() const;
15
16      static Donor parseLine(const std::string& line);
17
18  private:
19      static std::string trim(const std::string& str);
20  };
21
22 #endif // DONOR_H
```

Recuadro 3. Clase donante cruz roja

Semántica orientada a objetos en C++

La clase `Donor` representa a un donante en el Sistema de Gestión de Donantes de Sangre de la Cruz Roja. Cada objeto de esta clase contiene información clave como el ID del donante, su distrito, tipo de sangre, nombre, dirección y número de contacto. En la clase se adiciona la edad para cumplir con los requerimientos. Los métodos que permiten resolver los requisitos de la clase y permiten acceder a estos atributos, son: `getAge()`, `getBloodType()`, y `getDistrict()`. Además, la clase incluye un método estático `parseLine()` para convertir una línea de texto en un objeto `Donor`, facilitando la carga de información desde archivos. Para esta clase revisar el encapsulamiento tal como se solicita al final de esta guía.

Los métodos get se implementan con un return de la siguiente manera:

```
1  int Donor::getAge() const {
2      return age;
3  }
```

Recuadro 4. método get

En el contexto de programación orientada a objetos en C++, los métodos de una clase se utilizan a través de instancias de objetos, usando el operador `.`. Por ejemplo, si tenemos un objeto `donante` de la clase `Donor`, podemos acceder a sus métodos como `donante.getAge()` para obtener la edad del donante, o `donante.getBloodType()` para conocer su tipo de sangre. Este enfoque permite encapsular la lógica interna del objeto, de modo que solo interactuamos con sus funciones públicas sin necesidad de conocer cómo están implementadas. Esto asegura que la manipulación de los datos sea controlada y consistente. Así en la clase Reports se puede verificar el uso de los métodos de donante, ver el siguiente recuadro.

```
1  int Reports::donorTotal(const int &district, const int &age, const int &bloodType) const {
2      int total = 0;
3      for (const auto &d : auxDonor) {
4          if (d.district == district && d.age == age && d.bloodType == bloodType)
5              total++;
6      }
7      return total;
8  }
```

Recuadro 5. Uso del operador punto (.)

En C++, cuando se trabaja con punteros a objetos, los métodos de la clase se acceden utilizando el operador `->`. Este operador permite invocar funciones miembro a través de un puntero. Por ejemplo, si tenemos un puntero `Donor* donantePtr`, podríamos llamar a sus métodos usando `donantePtr->getAge()` para obtener la edad o `donantePtr->getBloodType()` para obtener el tipo de sangre. Este operador es esencial

Semántica orientada a objetos en C++

cuando se maneja la memoria dinámica o estructuras complejas, facilitando el acceso a los métodos sin necesidad de desreferenciar el puntero manualmente.

En C++, el puntero `this` es utilizado dentro de los métodos de una clase para referirse al objeto actual que está invocando ese método. Esto es útil cuando se necesita diferenciar entre los atributos de la clase y los parámetros del método que pueden tener el mismo nombre. Esto se verifica en la clase `Reports` en el método `loadDataBase`, ver el siguiente recuadro. En este caso `This` se refiere a que `auxDonor` es el que apunta al objeto en cuestión esto es el dato miembro de la clase `auxdonor` que es un objeto de donantes privado, diferenciado así `auxdonor` el parámetro que ingresa en el método `loadDataBase`. En otro caso dentro de la clase `Donor`, podríamos usar `this->age = age` para asignar el valor del parámetro `age` al atributo `age` del objeto actual (esto debido a que `age` no está encapsulado). `this` es un puntero implícito que siempre apunta al objeto en contexto, facilitando la manipulación de sus datos miembros desde dentro de los métodos.

```
1 void Reports::loadDataBase(const std::vector<Donor> &auxdonor) {  
2     this->auxDonor = auxdonor;  
3 }
```

Recuadro 6. Uso de `this`

Implementación del vector donantes

Un vector en programación es una estructura de datos que permite almacenar una colección de elementos del mismo tipo de forma secuencial. A diferencia de los arrays estáticos, los vectores tienen la capacidad de redimensionar automáticamente cuando es necesario agregar o eliminar elementos, lo que los hace mucho más flexibles [5]. Cada elemento en un vector se puede acceder mediante un índice, lo que facilita la manipulación de datos. Así como se pueden crear vectores de enteros o de strings se pueden crear vectores del tipo de datos que nosotros hemos creado como en este caso donantes. De esta manera todas las operaciones del vector se pueden llevar en este, creado por nuestra clase, para nuestro ejemplo la clase donante. Así, podemos saber el tamaño del vector con la función `size` o podemos limpiar un vector con la función `empty`.

El vector donantes es una estructura de datos fundamental dentro del sistema de gestión de donantes de sangre, encargada de almacenar los objetos de tipo `Donor`. Cada elemento del vector representa a un donante con sus respectivos atributos, como nombre, tipo de sangre, edad y procedencia. Este vector permite gestionar los datos de manera eficiente, ya que facilita la búsqueda, inserción y eliminación de donantes en el sistema. Además, al tratarse de un contenedor dinámico, el vector donantes se ajusta en tamaño según las necesidades del sistema, proporcionando flexibilidad en la administración de los datos sin necesidad de definir un límite fijo de almacenamiento. El vector de donantes se crea con la instrucción

Semántica orientada a objetos en C++

`std::vector<Donor> donors`; para lo cual es necesario incluir la librería `#include <vector>`, para más información sobre la librería `vector` `vec`: <https://cplusplus.com/reference/vector/vector/>

Finalmente, el método `loadDataBase` de la clase `Report` se utiliza para cargar los datos en el módulo de reportes. Recibe como parámetro un vector de donantes, el cual es obtenido a través del método de la clase `BloodDatabase` y su método `getDonors()`, que extrae el conjunto de donantes almacenados en el sistema. `database.getDonors()` devuelve un vector con todos los objetos `Donor`, que representan la información de cada donante, como su tipo de sangre, procedencia y edad. Una vez que los datos son obtenidos, `reporte.loadDataBase` los almacena localmente en `vector<Donor> auxDonor`, permitiendo que el sistema procese y genere las estadísticas requeridas. Este flujo garantiza que el módulo de reportes trabaje con la información actualizada al momento de solicitar el reporte en el sistema de gestión de donantes.

Taller 2

1. Abstracción: Revisar las abstracciones del Sistema de Gestión de Donantes de Sangre de la Cruz Roja. Se puede asegurar que las abstracciones permiten cumplir con los principios de responsabilidad única y abierto cerrado?
2. Encapsulamiento: Revisar el encapsulamiento de la clase `Donor`. Se puede asegurar que la información crítica sobre los donantes esté protegida y se manipule de forma segura dentro del sistema, evitando modificaciones indebidas y asegurando la integridad de los datos en el sistema?
3. Haga refactoring del proyecto, aplique los conceptos del punto 1 y 2 de este taller. Adicionalmente, entregue un proyecto con un código legible, entendible, que siga buenas prácticas de programación.
4. Hacer todo el proceso desde la “Tabla de requerimientos” hasta la codificación. Para la siguiente necesidad, el registro de unidades de sangre que dona una persona y la fecha de la donación.
5. Hacer todo el proceso desde la “Tabla de requerimientos” hasta la codificación. Para la siguiente necesidad, al finalizar el año se requiere hacer un reporte de cierre de año, en el cual se presente: el porcentaje de donantes por tipos de sangre y para cada una de las zonas del país.

Referencias

- [1] *Lógica y programación orientada a objetos: un enfoque basado en problemas*. Medellín: Tecnológico de Antioquia, 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] R. C. Martin, J. Grenning, S. Brown, y K. Henney, *Clean Architecture: a craftsman's*



Semántica orientada a objetos en C++

guide to software structure and design. en Robert C. Martin series. Boston Columbus Indianapolis New York San Francisco Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo: Prentice Hall, 2018.

- [4] M. Aniche, *Simple object-oriented design: create clean, maintainable applications*, [First edition]. Shelter Island, NY: Manning Publications, 2024.
- [5] B. Cyganek, *Introduction to programming with C++ for engineers*. Chichester, West Sussex, UK ; Hoboken, NJ, USA: Wiley-IEEE Press, 2020.
- [6] Y. Bugayenko, *Elegant objects. volume 1*, Version: 1.7. Palo Alto, California: Yegor Bugayenko, 9.