

Sistema de Gestión Polimórfica de Sensores IoT

1.0.0

Generado el Jueves, 30 de octubre de 2025 13:10:19 para Sistema de Gestión Polimórfica de Sensores IoT por Doxygen 1.14.0

Jueves, 30 de octubre de 2025 13:10:19

1 Índice jerárquico	2
1.1 Jerarquía de clases	2
2 Índice de clases	2
2.1 Lista de clases	2
3 Índice de archivos	3
3.1 Lista de archivos	3
4 Documentación de clases	3
4.1 Referencia de la clase ListaGestion	3
4.1.1 Descripción detallada	4
4.1.2 Documentación de constructores y destructores	4
4.1.3 Documentación de funciones miembro	5
4.1.4 Documentación de datos miembro	7
4.2 Referencia de la plantilla de la clase ListaSensor< T >	7
4.2.1 Descripción detallada	8
4.2.2 Documentación de constructores y destructores	8
4.2.3 Documentación de funciones miembro	9
4.2.4 Documentación de datos miembro	12
4.3 Referencia de la plantilla de la estructura Nodo< T >	12
4.3.1 Descripción detallada	13
4.3.2 Documentación de constructores y destructores	13
4.3.3 Documentación de datos miembro	14
4.4 Referencia de la estructura NodoSensor	14
4.4.1 Descripción detallada	15
4.4.2 Documentación de constructores y destructores	15
4.4.3 Documentación de datos miembro	15
4.5 Referencia de la clase SensorBase	16
4.5.1 Descripción detallada	16
4.5.2 Documentación de constructores y destructores	16
4.5.3 Documentación de funciones miembro	17
4.5.4 Documentación de datos miembro	18
4.6 Referencia de la clase SensorPresion	19
4.6.1 Descripción detallada	20
4.6.2 Documentación de constructores y destructores	20
4.6.3 Documentación de funciones miembro	20
4.6.4 Documentación de datos miembro	21
4.7 Referencia de la clase SensorTemperatura	22
4.7.1 Descripción detallada	23
4.7.2 Documentación de constructores y destructores	23
4.7.3 Documentación de funciones miembro	23
4.7.4 Documentación de datos miembro	24

4.8 Referencia de la clase SensorVibracion	25
4.8.1 Descripción detallada	26
4.8.2 Documentación de constructores y destructores	26
4.8.3 Documentación de funciones miembro	26
4.8.4 Documentación de datos miembro	27
4.9 Referencia de la clase SimuladorSerial	28
4.9.1 Descripción detallada	28
4.9.2 Documentación de constructores y destructores	28
4.9.3 Documentación de funciones miembro	28
4.9.4 Documentación de datos miembro	31
5 Documentación de archivos	31
5.1 Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores- iot/include/ListaGestion.h	31
5.2 ListaGestion.h	31
5.3 Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores- iot/include/ListaSensor.h	33
5.3.1 Descripción detallada	33
5.4 ListaSensor.h	34
5.5 Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores- iot/include/SensorBase.h	36
5.5.1 Descripción detallada	36
5.6 SensorBase.h	37
5.7 Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores- iot/include/SensorPresion.h	37
5.7.1 Descripción detallada	38
5.8 SensorPresion.h	38
5.9 Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores- iot/include/SensorTemperatura.h	39
5.9.1 Descripción detallada	39
5.10 SensorTemperatura.h	40
5.11 Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores- iot/include/SensorVibracion.h	41
5.11.1 Descripción detallada	41
5.12 SensorVibracion.h	42
5.13 Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores- iot/include/SimuladorSerial.h	43
5.13.1 Descripción detallada	43
5.14 SimuladorSerial.h	43
5.15 Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores- iot/README.md	45
5.16 Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores- iot/src/main.cpp	45
5.16.1 Descripción detallada	45
5.16.2 Documentación de funciones	46

1. Índice jerárquico

1.1. Jerarquía de clases

Este listado de herencia está ordenado de forma general pero no está en orden alfabético estricto:

ListaGestion	3
ListaSensor< T >	7
Nodo< T >	12
NodoSensor	14
SensorBase	16
SensorPresion	19
SensorTemperatura	22
SensorVibracion	25
SimuladorSerial	28

2. Índice de clases

2.1. Lista de clases

Lista de clases, estructuras, uniones e interfaces con breves descripciones:

ListaGestion	
Lista enlazada de sensores con gestión polimórfica	3
ListaSensor< T >	
Lista enlazada genérica con gestión manual de memoria	7
Nodo< T >	
Nodo de lista enlazada genérico	12
NodoSensor	
Nodo que almacena punteros a sensores (polimórfico)	14
SensorBase	
Clase abstracta que define la interfaz para todos los sensores	16
SensorPresion	
Sensor que mide presión atmosférica en hPa (tipo int)	19
SensorTemperatura	
Sensor que mide temperatura en °C (tipo float)	22

SensorVibracion

Sensor que mide intensidad de vibración (tipo int) 25

SimuladorSerial

Simula datos recibidos por puerto serial desde Arduino 28

3. Índice de archivos

3.1. Lista de archivos

Lista de todos los archivos con breves descripciones:

/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/ ListaGestion.h	
Lista polimórfica para gestionar sensores	31
/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/ ListaSensor.h	
Lista enlazada genérica para almacenar lecturas de sensores	33
/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/ SensorBase.h	
Clase base abstracta para todos los sensores	36
/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/ SensorPresion.h	
Sensor de presión atmosférica (valores enteros)	37
/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/ SensorTemperatura.h	
Sensor de temperatura (valores flotantes)	39
/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/ SensorVibracion.h	
Sensor de vibración (valores enteros)	41
/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/ SimuladorSerial.h	
Simulador de comunicación serial Arduino	43
/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/src/ main.cpp	
Programa principal SIMPLIFICADO del Sistema IoT	45

4. Documentación de clases

4.1. Referencia de la clase ListaGestion

Lista enlazada de sensores con gestión polimórfica.

```
#include <ListaGestion.h>
```

Diagrama de colaboración de ListaGestion:

Métodos públicos

- [ListaGestion](#) ()
Constructor.
- [~ListaGestion](#) ()
Destructor.
- void [agregarSensor](#) ([SensorBase](#) *sensor)
Agrega un sensor a la lista.
- [SensorBase](#) * [buscarPorId](#) (const char *id)
Busca un sensor por ID.
- void [procesarTodosSensores](#) ()
Procesa todos los sensores (polimórfico)
- void [mostrarTodos](#) () const
Muestra información de todos los sensores.
- int [getCantidad](#) () const
Obtiene la cantidad de sensores.

Métodos privados

- void [liberar](#) ()
Libera memoria de todos los sensores.

Atributos privados

- [NodoSensor](#) * [cabeza](#)
Primer nodo.
- int [cantidad](#)
Número de sensores.

4.1.1. Descripción detallada

Lista enlazada de sensores con gestión polimórfica.

Almacena punteros a [SensorBase](#), permitiendo procesamiento uniforme de diferentes tipos de sensores.

Definición en la línea [36](#) del archivo [ListaGestion.h](#).

4.1.2. Documentación de constructores y destructores

ListaGestion()

```
ListaGestion::ListaGestion () [inline]
```

Constructor.

Definición en la línea [60](#) del archivo [ListaGestion.h](#).

Hace referencia a [cabeza](#) y [cantidad](#).

~ListaGestion()

```
ListaGestion::~~ListaGestion () [inline]
```

Destructor.

Definición en la línea 65 del archivo [ListaGestion.h](#).

Hace referencia a [liberar\(\)](#).

Gráfico de llamadas de esta función:

4.1.3. Documentación de funciones miembro

agregarSensor()

```
void ListaGestion::agregarSensor (
    SensorBase * sensor) [inline]
```

Agrega un sensor a la lista.

Parámetros

<i>sensor</i>	Puntero al sensor (será propiedad de la lista)
---------------	--

Definición en la línea 73 del archivo [ListaGestion.h](#).

Hace referencia a [cabeza](#), [cantidad](#) y [NodoSensor::siguiente](#).

Referenciado por [main\(\)](#) y [procesarDatoArduino\(\)](#).

Gráfico de llamadas a esta función:

buscarPorId()

```
SensorBase * ListaGestion::buscarPorId (
    const char * id) [inline]
```

Busca un sensor por ID.

Parámetros

<i>id</i>	Identificador a buscar
-----------	------------------------

Devuelve

Puntero al sensor o nullptr si no existe

Definición en la línea 94 del archivo [ListaGestion.h](#).

Hace referencia a [cabeza](#), [SensorBase::getId\(\)](#), [NodoSensor::sensor](#) y [NodoSensor::siguiente](#).

Referenciado por [main\(\)](#) y [procesarDatoArduino\(\)](#).

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

getCantidad()

```
int ListaGestion::getCantidad () const [inline]
```

Obtiene la cantidad de sensores.

Devuelve

Número de sensores

Definición en la línea 149 del archivo [ListaGestion.h](#).

Hace referencia a [cantidad](#).

liberar()

```
void ListaGestion::liberar () [inline], [private]
```

Libera memoria de todos los sensores.

Definición en la línea 44 del archivo [ListaGestion.h](#).

Hace referencia a [cabeza](#), [cantidad](#), [NodoSensor::sensor](#) y [NodoSensor::siguiente](#).

Referenciado por [~ListaGestion\(\)](#).

Gráfico de llamadas a esta función:

mostrarTodos()

```
void ListaGestion::mostrarTodos () const [inline]
```

Muestra información de todos los sensores.

Definición en la línea 130 del archivo [ListaGestion.h](#).

Hace referencia a [cabeza](#), [cantidad](#), [SensorBase::imprimirInfo\(\)](#), [NodoSensor::sensor](#) y [NodoSensor::siguiente](#).

Referenciado por [main\(\)](#).

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

procesarTodosSensores()

```
void ListaGestion::procesarTodosSensores () [inline]
```

Procesa todos los sensores (polimórfico)

Llama a [procesarLectura\(\)](#) de cada sensor mediante despachado dinámico (virtual).

Definición en la línea 111 del archivo [ListaGestion.h](#).

Hace referencia a [cabeza](#), [cantidad](#), [SensorBase::getId\(\)](#), [SensorBase::procesarLectura\(\)](#), [NodoSensor::sensor](#) y [NodoSensor::siguiente](#).

Referenciado por [main\(\)](#).

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

4.1.4. Documentación de datos miembro

cabeza

```
NodoSensor* ListaGestion::cabeza [private]
```

Primer nodo.

Definición en la línea 38 del archivo [ListaGestion.h](#).

Referenciado por [agregarSensor\(\)](#), [buscarPorId\(\)](#), [liberar\(\)](#), [ListaGestion\(\)](#), [mostrarTodos\(\)](#) y [procesarTodosSensores\(\)](#).

cantidad

```
int ListaGestion::cantidad [private]
```

Número de sensores.

Definición en la línea 39 del archivo [ListaGestion.h](#).

Referenciado por [agregarSensor\(\)](#), [getCantidad\(\)](#), [liberar\(\)](#), [ListaGestion\(\)](#), [mostrarTodos\(\)](#) y [procesarTodosSensores\(\)](#).

La documentación de esta clase está generada del siguiente archivo:

- `/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/ListaGestion.h`

4.2. Referencia de la plantilla de la clase ListaSensor< T >

Lista enlazada genérica con gestión manual de memoria.

```
#include <ListaSensor.h>
```

Diagrama de herencia de ListaSensor< T >

Diagrama de colaboración de ListaSensor< T >:

Métodos públicos

- [ListaSensor](#) ()
Constructor por defecto.
- [~ListaSensor](#) ()
Destructor.
- [ListaSensor](#) (const [ListaSensor](#) &otra)
Constructor de copia (Regla de Tres)
- [ListaSensor](#) & [operator=](#) (const [ListaSensor](#) &otra)
Operador de asignación (Regla de Tres)
- void [agregar](#) (T valor)
Agrega un elemento al final.
- T [calcularPromedio](#) () const
Calcula el promedio de los valores.
- int [getCantidad](#) () const
Obtiene el número de elementos.
- void [imprimir](#) () const
Imprime todos los valores.
- bool [estaVacía](#) () const
Verifica si la lista está vacía.

Métodos privados

- void `liberar` ()
Libera toda la memoria de la lista.
- void `copiar` (const `ListaSensor` &otra)
Copia profunda de otra lista.

Atributos privados

- `Nodo`< T > * `cabeza`
Puntero al primer nodo.
- int `cantidad`
Número de elementos.

4.2.1. Descripción detallada

```
template<typename T>
class ListaSensor< T >
```

Lista enlazada genérica con gestión manual de memoria.

Parámetros de plantilla

<code>T</code>	Tipo de dato a almacenar (int, float, etc.)
----------------	---

Implementa la Regla de Tres (constructor copia, operador=, destructor) para gestión correcta de memoria dinámica.

Definición en la línea 39 del archivo `ListaSensor.h`.

4.2.2. Documentación de constructores y destructores

ListaSensor() [1/2]

```
template<typename T>
ListaSensor< T >::ListaSensor () [inline]
```

Constructor por defecto.

Definición en la línea 86 del archivo `ListaSensor.h`.

Hace referencia a `cabeza` y `cantidad`.

Referenciado por `copiar()`, `ListaSensor()` y `operator=()`.

Gráfico de llamadas a esta función:

~ListaSensor()

```
template<typename T>
ListaSensor< T >::~~ListaSensor () [inline]
```

Destructor.

Definición en la línea 91 del archivo [ListaSensor.h](#).

Hace referencia a [liberar\(\)](#).

Gráfico de llamadas de esta función:

ListaSensor() [2/2]

```
template<typename T>
ListaSensor< T >::ListaSensor (
    const ListaSensor< T > & otra) [inline]
```

Constructor de copia (Regla de Tres)

Parámetros

<i>otra</i>	Lista a copiar
-------------	----------------

Definición en la línea 99 del archivo [ListaSensor.h](#).

Hace referencia a [cabeza](#), [cantidad](#), [copiar\(\)](#) y [ListaSensor\(\)](#).

Gráfico de llamadas de esta función:

4.2.3. Documentación de funciones miembro

agregar()

```
template<typename T>
void ListaSensor< T >::agregar (
    T valor) [inline]
```

Agrega un elemento al final.

Parámetros

<i>valor</i>	Valor a agregar
--------------	-----------------

Definición en la línea 120 del archivo [ListaSensor.h](#).

Hace referencia a [cabeza](#), [cantidad](#) y [Nodo< T >::siguiente](#).

calcularPromedio()

```
template<typename T>
T ListaSensor< T >::calcularPromedio () const [inline]
```

Calcula el promedio de los valores.

Devuelve

Promedio (tipo T)

Definición en la línea 140 del archivo [ListaSensor.h](#).

Hace referencia a [cabeza](#), [cantidad](#), [Nodo< T >::dato](#) y [Nodo< T >::siguiente](#).

copiar()

```
template<typename T>
void ListaSensor< T >::copiar (
    const ListaSensor< T > & otra) [inline], [private]
```

Copia profunda de otra lista.

Parámetros

<i>otra</i>	Lista a copiar
-------------	----------------

Definición en la línea 62 del archivo [ListaSensor.h](#).

Hace referencia a [cabeza](#), [cantidad](#), [Nodo< T >::dato](#), [ListaSensor\(\)](#) y [Nodo< T >::siguiente](#).

Referenciado por [ListaSensor\(\)](#) y [operator=\(\)](#).

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

estaVacia()

```
template<typename T>
bool ListaSensor< T >::estaVacia () const [inline]
```

Verifica si la lista está vacía.

Devuelve

true si está vacía

Definición en la línea 182 del archivo [ListaSensor.h](#).

Hace referencia a [cabeza](#).

getCantidad()

```
template<typename T>
int ListaSensor< T >::getCantidad () const [inline]
```

Obtiene el número de elementos.

Devuelve

Cantidad de elementos

Definición en la línea 158 del archivo [ListaSensor.h](#).

Hace referencia a [cantidad](#).

imprimir()

```
template<typename T>
void ListaSensor< T >::imprimir () const [inline]
```

Imprime todos los valores.

Definición en la línea 165 del archivo [ListaSensor.h](#).

Hace referencia a [cabeza](#), [Nodo< T >::dato](#) y [Nodo< T >::siguiente](#).

liberar()

```
template<typename T>
void ListaSensor< T >::liberar () [inline], [private]
```

Libera toda la memoria de la lista.

Definición en la línea 47 del archivo [ListaSensor.h](#).

Hace referencia a [cabeza](#), [cantidad](#) y [Nodo< T >::siguiente](#).

Referenciado por [operator=\(\)](#) y [~ListaSensor\(\)](#).

Gráfico de llamadas a esta función:

operator=()

```
template<typename T>
ListaSensor & ListaSensor< T >::operator= (
    const ListaSensor< T > & otra) [inline]
```

Operador de asignación (Regla de Tres)

Parámetros

<i>otra</i>	Lista a asignar
-------------	-----------------

Devuelve

Referencia a esta lista

Definición en la línea 108 del archivo [ListaSensor.h](#).

Hace referencia a [copiar\(\)](#), [liberar\(\)](#) y [ListaSensor\(\)](#).

Gráfico de llamadas de esta función:

4.2.4. Documentación de datos miembro**cabeza**

```
template<typename T>
Nodo<T>* ListaSensor< T >::cabeza [private]
```

Puntero al primer nodo.

Definición en la línea 41 del archivo [ListaSensor.h](#).

Referenciado por [agregar\(\)](#), [calcularPromedio\(\)](#), [copiar\(\)](#), [estaVacia\(\)](#), [imprimir\(\)](#), [liberar\(\)](#), [ListaSensor\(\)](#) y [ListaSensor\(\)](#).

cantidad

```
template<typename T>
int ListaSensor< T >::cantidad [private]
```

Número de elementos.

Definición en la línea 42 del archivo [ListaSensor.h](#).

Referenciado por [agregar\(\)](#), [calcularPromedio\(\)](#), [copiar\(\)](#), [getCantidad\(\)](#), [liberar\(\)](#), [ListaSensor\(\)](#) y [ListaSensor\(\)](#).

La documentación de esta clase está generada del siguiente archivo:

- [/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/ListaSensor.h](#)

4.3. Referencia de la plantilla de la estructura `Nodo< T >`

[Nodo](#) de lista enlazada genérico.

```
#include <ListaSensor.h>
```

Diagrama de herencia de `Nodo< T >`

Diagrama de colaboración de `Nodo< T >`:

Métodos públicos

- `Nodo` (T valor)

Constructor.

Atributos públicos

- T `dato`

Dato almacenado.

- `Nodo` * `siguiente`

Puntero al siguiente nodo.

4.3.1. Descripción detallada

```
template<typename T>
struct Nodo< T >
```

`Nodo` de lista enlazada genérico.

Parámetros de plantilla

<code>T</code>	Tipo de dato a almacenar
----------------	--------------------------

Definición en la línea 19 del archivo [ListaSensor.h](#).

4.3.2. Documentación de constructores y destructores

`Nodo()`

```
template<typename T>
Nodo< T >::Nodo (
    T valor) [inline]
```

Constructor.

Parámetros

<code>valor</code>	Valor a almacenar
--------------------	-------------------

Definición en la línea 27 del archivo [ListaSensor.h](#).

Hace referencia a `dato` y `siguiente`.

4.3.3. Documentación de datos miembro

dato

```
template<typename T>
T Nodo< T >::dato
```

Dato almacenado.

Definición en la línea 20 del archivo [ListaSensor.h](#).

Referenciado por [ListaSensor< T >::calcularPromedio\(\)](#), [ListaSensor< T >::copiar\(\)](#), [ListaSensor< T >::imprimir\(\)](#) y [Nodo\(\)](#).

siguiente

```
template<typename T>
Nodo* Nodo< T >::siguiente
```

Puntero al siguiente nodo.

Definición en la línea 21 del archivo [ListaSensor.h](#).

Referenciado por [ListaSensor< T >::agregar\(\)](#), [ListaSensor< T >::calcularPromedio\(\)](#), [ListaSensor< T >::copiar\(\)](#), [ListaSensor< T >::imprimir\(\)](#), [ListaSensor< T >::liberar\(\)](#) y [Nodo\(\)](#).

La documentación de esta estructura está generada del siguiente archivo:

- [/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/ListaSensor.h](#)

4.4. Referencia de la estructura NodoSensor

[Nodo](#) que almacena punteros a sensores (polimórfico)

```
#include <ListaGestion.h>
```

Diagrama de colaboración de [NodoSensor](#):

Métodos públicos

- [NodoSensor](#) ([SensorBase](#) *s)
Constructor.

Atributos públicos

- [SensorBase](#) * [sensor](#)
Puntero polimórfico a sensor.
- [NodoSensor](#) * [siguiente](#)
Siguiente nodo.

4.4.1. Descripción detallada

`Nodo` que almacena punteros a sensores (polimórfico)

Definición en la línea 18 del archivo [ListaGestion.h](#).

4.4.2. Documentación de constructores y destructores

`NodoSensor()`

```
NodoSensor::NodoSensor (  
    SensorBase * s) [inline]
```

Constructor.

Parámetros

s	Puntero al sensor
---	-------------------

Definición en la línea 26 del archivo [ListaGestion.h](#).

Hace referencia a [sensor](#) y [siguiente](#).

4.4.3. Documentación de datos miembro

`sensor`

```
SensorBase* NodoSensor::sensor
```

Puntero polimórfico a sensor.

Definición en la línea 19 del archivo [ListaGestion.h](#).

Referenciado por [ListaGestion::buscarPorId\(\)](#), [ListaGestion::liberar\(\)](#), [ListaGestion::mostrarTodos\(\)](#), [NodoSensor\(\)](#) y [ListaGestion::procesarTodosSensores\(\)](#).

`siguiente`

```
NodoSensor* NodoSensor::siguiente
```

Siguiente nodo.

Definición en la línea 20 del archivo [ListaGestion.h](#).

Referenciado por [ListaGestion::agregarSensor\(\)](#), [ListaGestion::buscarPorId\(\)](#), [ListaGestion::liberar\(\)](#), [ListaGestion::mostrarTodos\(\)](#), [NodoSensor\(\)](#) y [ListaGestion::procesarTodosSensores\(\)](#).

La documentación de esta estructura está generada del siguiente archivo:

- `/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/ListaGestion.h`

4.5. Referencia de la clase SensorBase

Clase abstracta que define la interfaz para todos los sensores.

```
#include <SensorBase.h>
```

Diagrama de herencia de SensorBase

Diagrama de colaboración de SensorBase:

Métodos públicos

- [SensorBase](#) (const char *id, const char *ubi)
Constructor parametrizado.
- virtual [~SensorBase](#) ()
Destructor virtual (crítico para polimorfismo)
- virtual void [procesarLectura](#) ()=0
Procesa una lectura del sensor (método virtual puro)
- virtual void [imprimirInfo](#) () const =0
Imprime información del sensor (método virtual puro)
- const char * [getId](#) () const
Obtiene el ID del sensor.
- const char * [getUbicacion](#) () const
Obtiene la ubicación del sensor.

Atributos protegidos

- char * [id](#)
Identificador único del sensor.
- char * [ubicacion](#)
Ubicación física del sensor.

4.5.1. Descripción detallada

Clase abstracta que define la interfaz para todos los sensores.

Esta clase utiliza polimorfismo para permitir procesamiento uniforme de diferentes tipos de sensores a través de métodos virtuales puros.

Definición en la línea 21 del archivo [SensorBase.h](#).

4.5.2. Documentación de constructores y destructores

SensorBase()

```
SensorBase::SensorBase (
    const char * id,
    const char * ubi) [inline]
```

Constructor parametrizado.

Parámetros

<i>id</i>	Identificador del sensor
<i>ubi</i>	Ubicación del sensor

Definición en la línea 32 del archivo [SensorBase.h](#).

Referenciado por [SensorPresion::SensorPresion\(\)](#), [SensorTemperatura::SensorTemperatura\(\)](#) y [SensorVibracion::SensorVibracion\(\)](#).

Gráfico de llamadas a esta función:

~SensorBase()

```
virtual SensorBase::~~SensorBase () [inline], [virtual]
```

Destructor virtual (crítico para polimorfismo)

Definición en la línea 43 del archivo [SensorBase.h](#).

Hace referencia a [id](#) y [ubicacion](#).

4.5.3. Documentación de funciones miembro**getId()**

```
const char * SensorBase::getId () const [inline]
```

Obtiene el ID del sensor.

Devuelve

Puntero al identificador

Definición en la línea 62 del archivo [SensorBase.h](#).

Hace referencia a [id](#).

Referenciado por [ListaGestion::buscarPorId\(\)](#) y [ListaGestion::procesarTodosSensores\(\)](#).

Gráfico de llamadas a esta función:

getUbicacion()

```
const char * SensorBase::getUbicacion () const [inline]
```

Obtiene la ubicación del sensor.

Devuelve

Puntero a la ubicación

Definición en la línea 68 del archivo [SensorBase.h](#).

Hace referencia a [ubicacion](#).

imprimirInfo()

```
virtual void SensorBase::imprimirInfo () const [pure virtual]
```

Imprime información del sensor (método virtual puro)

Implementado en [SensorPresion](#), [SensorTemperatura](#) y [SensorVibracion](#).

Referenciado por [ListaGestion::mostrarTodos\(\)](#).

Gráfico de llamadas a esta función:

procesarLectura()

```
virtual void SensorBase::procesarLectura () [pure virtual]
```

Procesa una lectura del sensor (método virtual puro)

Implementado en [SensorPresion](#), [SensorTemperatura](#) y [SensorVibracion](#).

Referenciado por [ListaGestion::procesarTodosSensores\(\)](#).

Gráfico de llamadas a esta función:

4.5.4. Documentación de datos miembro

id

```
char* SensorBase::id [protected]
```

Identificador único del sensor.

Definición en la línea [23](#) del archivo [SensorBase.h](#).

Referenciado por [getId\(\)](#), [SensorPresion::SensorPresion\(\)](#), [SensorTemperatura::SensorTemperatura\(\)](#), [SensorVibracion::SensorVibracion\(\)](#) y [~SensorBase\(\)](#).

ubicacion

```
char* SensorBase::ubicacion [protected]
```

Ubicación física del sensor.

Definición en la línea [24](#) del archivo [SensorBase.h](#).

Referenciado por [getUbicacion\(\)](#), [SensorPresion::imprimirInfo\(\)](#), [SensorTemperatura::imprimirInfo\(\)](#), [SensorVibracion::imprimirInfo\(\)](#) y [~SensorBase\(\)](#).

La documentación de esta clase está generada del siguiente archivo:

- [/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/SensorBase.h](#)

4.6. Referencia de la clase SensorPresion

Sensor que mide presión atmosférica en hPa (tipo int)

```
#include <SensorPresion.h>
```

Diagrama de herencia de SensorPresion

Diagrama de colaboración de SensorPresion:

Métodos públicos

- **SensorPresion** (const char *id, const char *ubi)
Constructor.
- void **agregarLectura** (int valor)
Agrega una lectura de presión.
- void **procesarLectura** () override
Procesa las lecturas de presión.
- void **imprimirInfo** () const override
Imprime información del sensor.
- const **ListaSensor**< int > & **getLecturas** () const
Obtiene la lista de lecturas.

Métodos públicos heredados de **SensorBase**

- **SensorBase** (const char *id, const char *ubi)
Constructor parametrizado.
- virtual **~SensorBase** ()
Destructor virtual (crítico para polimorfismo)
- const char * **getId** () const
Obtiene el ID del sensor.
- const char * **getUbicacion** () const
Obtiene la ubicación del sensor.

Atributos privados

- **ListaSensor**< int > **lecturas**
Lista de lecturas de presión.

Otros miembros heredados

Atributos protegidos heredados de **SensorBase**

- char * **id**
Identificador único del sensor.
- char * **ubicacion**
Ubicación física del sensor.

4.6.1. Descripción detallada

Sensor que mide presión atmosférica en hPa (tipo int)

Hereda de [SensorBase](#) e implementa procesamiento específico para datos de presión.

Definición en la línea 21 del archivo [SensorPresion.h](#).

4.6.2. Documentación de constructores y destructores

SensorPresion()

```
SensorPresion::SensorPresion (  
    const char * id,  
    const char * ubi) [inline]
```

Constructor.

Parámetros

<i>id</i>	Identificador del sensor
<i>ubi</i>	Ubicación del sensor

Definición en la línea 31 del archivo [SensorPresion.h](#).

Hace referencia a [SensorBase::id](#) y [SensorBase::SensorBase\(\)](#).

Gráfico de llamadas de esta función:

4.6.3. Documentación de funciones miembro

agregarLectura()

```
void SensorPresion::agregarLectura (  
    int valor) [inline]
```

Agrega una lectura de presión.

Parámetros

<i>valor</i>	Presión en hPa
--------------	----------------

Definición en la línea 38 del archivo [SensorPresion.h](#).

Hace referencia a [lecturas](#).

Referenciado por [main\(\)](#) y [procesarDatoArduino\(\)](#).

Gráfico de llamadas a esta función:

getLecturas()

```
const ListaSensor< int > & SensorPresion::getLecturas () const [inline]
```

Obtiene la lista de lecturas.

Devuelve

Referencia a la lista

Definición en la línea 81 del archivo [SensorPresion.h](#).

Hace referencia a [lecturas](#).

imprimirInfo()

```
void SensorPresion::imprimirInfo () const [inline], [override], [virtual]
```

Imprime información del sensor.

Implementa [SensorBase](#).

Definición en la línea 68 del archivo [SensorPresion.h](#).

Hace referencia a [lecturas](#) y [SensorBase::ubicacion](#).

procesarLectura()

```
void SensorPresion::procesarLectura () [inline], [override], [virtual]
```

Procesa las lecturas de presión.

Calcula promedio y verifica límites (980-1050 hPa)

Implementa [SensorBase](#).

Definición en la línea 47 del archivo [SensorPresion.h](#).

Hace referencia a [lecturas](#).

4.6.4. Documentación de datos miembro**lecturas**

```
ListaSensor<int> SensorPresion::lecturas [private]
```

Lista de lecturas de presión.

Definición en la línea 23 del archivo [SensorPresion.h](#).

Referenciado por [agregarLectura\(\)](#), [getLecturas\(\)](#), [imprimirInfo\(\)](#) y [procesarLectura\(\)](#).

La documentación de esta clase está generada del siguiente archivo:

- [/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/SensorPresion.h](#)

4.7. Referencia de la clase SensorTemperatura

Sensor que mide temperatura en °C (tipo float)

```
#include <SensorTemperatura.h>
```

Diagrama de herencia de SensorTemperatura

Diagrama de colaboración de SensorTemperatura:

Métodos públicos

- **SensorTemperatura** (const char *id, const char *ubi)
Constructor.
- void **agregarLectura** (float valor)
Agrega una lectura de temperatura.
- void **procesarLectura** () override
Procesa las lecturas de temperatura.
- void **imprimirInfo** () const override
Imprime información del sensor.
- const **ListaSensor**< float > & **getLecturas** () const
Obtiene la lista de lecturas.

Métodos públicos heredados de **SensorBase**

- **SensorBase** (const char *id, const char *ubi)
Constructor parametrizado.
- virtual **~SensorBase** ()
Destructor virtual (crítico para polimorfismo)
- const char * **getId** () const
Obtiene el ID del sensor.
- const char * **getUbicacion** () const
Obtiene la ubicación del sensor.

Atributos privados

- **ListaSensor**< float > **lecturas**
Lista de lecturas de temperatura.

Otros miembros heredados

Atributos protegidos heredados de **SensorBase**

- char * **id**
Identificador único del sensor.
- char * **ubicacion**
Ubicación física del sensor.

4.7.1. Descripción detallada

Sensor que mide temperatura en °C (tipo float)

Hereda de [SensorBase](#) e implementa procesamiento específico para datos de temperatura.

Definición en la línea 21 del archivo [SensorTemperatura.h](#).

4.7.2. Documentación de constructores y destructores

SensorTemperatura()

```
SensorTemperatura::SensorTemperatura (
    const char * id,
    const char * ubi) [inline]
```

Constructor.

Parámetros

<i>id</i>	Identificador del sensor
<i>ubi</i>	Ubicación del sensor

Definición en la línea 31 del archivo [SensorTemperatura.h](#).

Hace referencia a [SensorBase::id](#) y [SensorBase::SensorBase\(\)](#).

Gráfico de llamadas de esta función:

4.7.3. Documentación de funciones miembro

agregarLectura()

```
void SensorTemperatura::agregarLectura (
    float valor) [inline]
```

Agrega una lectura de temperatura.

Parámetros

<i>valor</i>	Temperatura en °C
--------------	-------------------

Definición en la línea 38 del archivo [SensorTemperatura.h](#).

Hace referencia a [lecturas](#).

Referenciado por [main\(\)](#) y [procesarDatoArduino\(\)](#).

Gráfico de llamadas a esta función:

getLecturas()

```
const ListaSensor< float > & SensorTemperatura::getLecturas () const [inline]
```

Obtiene la lista de lecturas.

Devuelve

Referencia a la lista

Definición en la línea 81 del archivo [SensorTemperatura.h](#).

Hace referencia a [lecturas](#).

imprimirInfo()

```
void SensorTemperatura::imprimirInfo () const [inline], [override], [virtual]
```

Imprime información del sensor.

Implementa [SensorBase](#).

Definición en la línea 68 del archivo [SensorTemperatura.h](#).

Hace referencia a [lecturas](#) y [SensorBase::ubicacion](#).

procesarLectura()

```
void SensorTemperatura::procesarLectura () [inline], [override], [virtual]
```

Procesa las lecturas de temperatura.

Calcula promedio y verifica límites (15-30°C)

Implementa [SensorBase](#).

Definición en la línea 47 del archivo [SensorTemperatura.h](#).

Hace referencia a [lecturas](#).

4.7.4. Documentación de datos miembro

lecturas

```
ListaSensor<float> SensorTemperatura::lecturas [private]
```

Lista de lecturas de temperatura.

Definición en la línea 23 del archivo [SensorTemperatura.h](#).

Referenciado por [agregarLectura\(\)](#), [getLecturas\(\)](#), [imprimirInfo\(\)](#) y [procesarLectura\(\)](#).

La documentación de esta clase está generada del siguiente archivo:

- [/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/SensorTemperatura.h](#)

4.8. Referencia de la clase SensorVibracion

Sensor que mide intensidad de vibración (tipo int)

```
#include <SensorVibracion.h>
```

Diagrama de herencia de SensorVibracion

Diagrama de colaboración de SensorVibracion:

Métodos públicos

- **SensorVibracion** (const char *id, const char *ubi)
Constructor.
- void **agregarLectura** (int valor)
Agrega una lectura de vibración.
- void **procesarLectura** () override
Procesa las lecturas de vibración.
- void **imprimirInfo** () const override
Imprime información del sensor.
- const **ListaSensor**< int > & **getLecturas** () const
Obtiene la lista de lecturas.

Métodos públicos heredados de **SensorBase**

- **SensorBase** (const char *id, const char *ubi)
Constructor parametrizado.
- virtual **~SensorBase** ()
Destructor virtual (crítico para polimorfismo)
- const char * **getId** () const
Obtiene el ID del sensor.
- const char * **getUbicacion** () const
Obtiene la ubicación del sensor.

Atributos privados

- **ListaSensor**< int > **lecturas**
Lista de lecturas de vibración.

Otros miembros heredados

Atributos protegidos heredados de **SensorBase**

- char * **id**
Identificador único del sensor.
- char * **ubicacion**
Ubicación física del sensor.

4.8.1. Descripción detallada

Sensor que mide intensidad de vibración (tipo int)

Hereda de [SensorBase](#) e implementa procesamiento específico para datos de vibración/aceleración.

Definición en la línea 21 del archivo [SensorVibracion.h](#).

4.8.2. Documentación de constructores y destructores

SensorVibracion()

```
SensorVibracion::SensorVibracion (
    const char * id,
    const char * ubi) [inline]
```

Constructor.

Parámetros

<i>id</i>	Identificador del sensor
<i>ubi</i>	Ubicación del sensor

Definición en la línea 31 del archivo [SensorVibracion.h](#).

Hace referencia a [SensorBase::id](#) y [SensorBase::SensorBase\(\)](#).

Gráfico de llamadas de esta función:

4.8.3. Documentación de funciones miembro

agregarLectura()

```
void SensorVibracion::agregarLectura (
    int valor) [inline]
```

Agrega una lectura de vibración.

Parámetros

<i>valor</i>	Intensidad (0-100)
--------------	--------------------

Definición en la línea 38 del archivo [SensorVibracion.h](#).

Hace referencia a [lecturas](#).

Referenciado por [main\(\)](#) y [procesarDatoArduino\(\)](#).

Gráfico de llamadas a esta función:

getLecturas()

```
const ListaSensor< int > & SensorVibracion::getLecturas () const [inline]
```

Obtiene la lista de lecturas.

Devuelve

Referencia a la lista

Definición en la línea 81 del archivo [SensorVibracion.h](#).

Hace referencia a [lecturas](#).

imprimirInfo()

```
void SensorVibracion::imprimirInfo () const [inline], [override], [virtual]
```

Imprime información del sensor.

Implementa [SensorBase](#).

Definición en la línea 68 del archivo [SensorVibracion.h](#).

Hace referencia a [lecturas](#) y [SensorBase::ubicacion](#).

procesarLectura()

```
void SensorVibracion::procesarLectura () [inline], [override], [virtual]
```

Procesa las lecturas de vibración.

Calcula promedio y verifica niveles de alerta (0-100)

Implementa [SensorBase](#).

Definición en la línea 47 del archivo [SensorVibracion.h](#).

Hace referencia a [lecturas](#).

4.8.4. Documentación de datos miembro**lecturas**

```
ListaSensor<int> SensorVibracion::lecturas [private]
```

Lista de lecturas de vibración.

Definición en la línea 23 del archivo [SensorVibracion.h](#).

Referenciado por [agregarLectura\(\)](#), [getLecturas\(\)](#), [imprimirInfo\(\)](#) y [procesarLectura\(\)](#).

La documentación de esta clase está generada del siguiente archivo:

- [/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/SensorVibracion.h](#)

4.9. Referencia de la clase SimuladorSerial

Simula datos recibidos por puerto serial desde Arduino.

```
#include <SimuladorSerial.h>
```

Diagrama de colaboración de SimuladorSerial:

Métodos públicos

- `SimuladorSerial ()`
Constructor.
- `void inicializar ()`
Inicializa el generador de números aleatorios.
- `void generarTemperatura (char *buffer, int tam)`
Genera una lectura de temperatura.
- `void generarPresion (char *buffer, int tam)`
Genera una lectura de presión.
- `void generarVibracion (char *buffer, int tam)`
Genera una lectura de vibración.
- `void generarLecturaAleatoria (char *buffer, int tam)`
Genera una lectura aleatoria de cualquier tipo.
- `void simularRecepcion (char *buffer, int tam, int numLecturas=5)`
Simula recepción de datos del Arduino.

Atributos privados

- `bool inicializado`
Estado de inicialización.

4.9.1. Descripción detallada

Simula datos recibidos por puerto serial desde Arduino.

Genera datos aleatorios en formato "TIPO:ID:VALOR" simulando el comportamiento del sketch Arduino.

Definición en la línea 23 del archivo [SimuladorSerial.h](#).

4.9.2. Documentación de constructores y destructores

SimuladorSerial()

```
SimuladorSerial::SimuladorSerial () [inline]
```

Constructor.

Definición en la línea 31 del archivo [SimuladorSerial.h](#).

Hace referencia a [inicializado](#).

4.9.3. Documentación de funciones miembro

generarLecturaAleatoria()

```
void SimuladorSerial::generarLecturaAleatoria (
    char * buffer,
    int tam) [inline]
```

Genera una lectura aleatoria de cualquier tipo.

Parámetros

<i>buffer</i>	Buffer donde escribir el dato
<i>tam</i>	Tamaño del buffer

Definición en la línea 78 del archivo [SimuladorSerial.h](#).

Hace referencia a [generarPresion\(\)](#), [generarTemperatura\(\)](#) y [generarVibracion\(\)](#).

Referenciado por [main\(\)](#) y [simularRecepcion\(\)](#).

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

generarPresion()

```
void SimuladorSerial::generarPresion (
    char * buffer,
    int tam) [inline]
```

Genera una lectura de presión.

Parámetros

<i>buffer</i>	Buffer donde escribir el dato
<i>tam</i>	Tamaño del buffer

Definición en la línea 58 del archivo [SimuladorSerial.h](#).

Referenciado por [generarLecturaAleatoria\(\)](#).

Gráfico de llamadas a esta función:

generarTemperatura()

```
void SimuladorSerial::generarTemperatura (
    char * buffer,
    int tam) [inline]
```

Genera una lectura de temperatura.

Parámetros

<i>buffer</i>	Buffer donde escribir el dato
<i>tam</i>	Tamaño del buffer

Definición en la línea 48 del archivo [SimuladorSerial.h](#).

Referenciado por [generarLecturaAleatoria\(\)](#).

Gráfico de llamadas a esta función:

generarVibracion()

```
void SimuladorSerial::generarVibracion (
    char * buffer,
    int tam) [inline]
```

Genera una lectura de vibración.

Parámetros

<i>buffer</i>	Buffer donde escribir el dato
<i>tam</i>	Tamaño del buffer

Definición en la línea [68](#) del archivo [SimuladorSerial.h](#).

Referenciado por [generarLecturaAleatoria\(\)](#).

Gráfico de llamadas a esta función:

inicializar()

```
void SimuladorSerial::inicializar () [inline]
```

Inicializa el generador de números aleatorios.

Definición en la línea [36](#) del archivo [SimuladorSerial.h](#).

Hace referencia a [inicializado](#).

Referenciado por [main\(\)](#) y [simularRecepcion\(\)](#).

Gráfico de llamadas a esta función:

simularRecepcion()

```
void SimuladorSerial::simularRecepcion (  
    char * buffer,  
    int tam,  
    int numLecturas = 5) [inline]
```

Simula recepción de datos del Arduino.

Parámetros

<i>buffer</i>	Buffer donde se escribirán los datos
<i>tam</i>	Tamaño del buffer
<i>numLecturas</i>	Número de lecturas a generar

Definición en la línea [100](#) del archivo [SimuladorSerial.h](#).

Hace referencia a [generarLecturaAleatoria\(\)](#) y [inicializar\(\)](#).

Gráfico de llamadas de esta función:

4.9.4. Documentación de datos miembro

inicializado

```
bool SimuladorSerial::inicializado [private]
```

Estado de inicialización.

Definición en la línea 25 del archivo [SimuladorSerial.h](#).

Referenciado por [inicializar\(\)](#) y [SimuladorSerial\(\)](#).

La documentación de esta clase está generada del siguiente archivo:

- [/home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/SimuladorSerial.h](#)

5. Documentación de archivos

5.1. Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/ListaGestion.h

Lista polimórfica para gestionar sensores.

```
#include "SensorBase.h"
#include <iostream>
```

Gráfico de dependencias incluidas en ListaGestion.h:

5.2. ListaGestion.h

[Ir a la documentación de este archivo.](#)

```
00001 /**
00002  * @file ListaGestion.h
00003  * @brief Lista polimórfica para gestionar sensores
00004  * @author Carlos Vargas
00005  * @date 30 de octubre de 2025
00006  */
00007
00008 #ifndef LISTA_GESTION_H
00009 #define LISTA_GESTION_H
00010
00011 #include "SensorBase.h"
00012 #include <iostream>
00013
00014 /**
00015  * @struct NodoSensor
00016  * @brief Nodo que almacena punteros a sensores (polimórfico)
00017  */
00018 struct NodoSensor {
00019     SensorBase* sensor;    ///< Puntero polimórfico a sensor
00020     NodoSensor* siguiente; ///< Siguiente nodo
00021
00022     /**
00023      * @brief Constructor
00024      * @param s Puntero al sensor
00025      */
00026     NodoSensor(SensorBase* s) : sensor(s), siguiente(nullptr) {}
00027 };
00028
00029 /**
00030  * @class ListaGestion
00031  * @brief Lista enlazada de sensores con gestión polimórfica
00032  */
```

```

00033  * Almacena punteros a SensorBase, permitiendo procesamiento
00034  * uniforme de diferentes tipos de sensores.
00035  */
00036  class ListaGestion {
00037  private:
00038      NodoSensor* cabeza;    ///< Primer nodo
00039      int cantidad;          ///< Número de sensores
00040
00041      /**
00042       * @brief Libera memoria de todos los sensores
00043       */
00044      void liberar() {
00045          NodoSensor* actual = cabeza;
00046          while (actual != nullptr) {
00047              NodoSensor* temp = actual;
00048              actual = actual->siguiente;
00049              delete temp->sensor; // Llama al destructor virtual
00050              delete temp;
00051          }
00052          cabeza = nullptr;
00053          cantidad = 0;
00054      }
00055
00056  public:
00057      /**
00058       * @brief Constructor
00059       */
00060      ListaGestion() : cabeza(nullptr), cantidad(0) {}
00061
00062      /**
00063       * @brief Destructor
00064       */
00065      ~ListaGestion() {
00066          liberar();
00067      }
00068
00069      /**
00070       * @brief Agrega un sensor a la lista
00071       * @param sensor Puntero al sensor (será propiedad de la lista)
00072       */
00073      void agregarSensor(SensorBase* sensor) {
00074          NodoSensor* nuevo = new NodoSensor(sensor);
00075
00076          if (cabeza == nullptr) {
00077              cabeza = nuevo;
00078          } else {
00079              NodoSensor* actual = cabeza;
00080              while (actual->siguiente != nullptr) {
00081                  actual = actual->siguiente;
00082              }
00083              actual->siguiente = nuevo;
00084          }
00085
00086          cantidad++;
00087      }
00088
00089      /**
00090       * @brief Busca un sensor por ID
00091       * @param id Identificador a buscar
00092       * @return Puntero al sensor o nullptr si no existe
00093       */
00094      SensorBase* buscarPorId(const char* id) {
00095          NodoSensor* actual = cabeza;
00096          while (actual != nullptr) {
00097              if (strcmp(actual->sensor->getId(), id) == 0) {
00098                  return actual->sensor;
00099              }
00100              actual = actual->siguiente;
00101          }
00102          return nullptr;
00103      }
00104
00105      /**
00106       * @brief Procesa todos los sensores (polimórfico)
00107       *
00108       * Llama a procesarLectura() de cada sensor mediante
00109       * despacho dinámico (virtual).
00110       */
00111      void procesarTodosSensores() {
00112          if (cantidad == 0) {
00113              std::cout << "\nNo hay sensores registrados" << std::endl;
00114              return;
00115          }
00116
00117          std::cout << "\n=== Procesando " << cantidad << " sensores ===" << std::endl;
00118
00119          NodoSensor* actual = cabeza;

```

```

00120         while (actual != nullptr) {
00121             std::cout << "\nSensor: " << actual->sensor->getId() << std::endl;
00122             actual->sensor->procesarLectura(); // Llamada polimórfica
00123             actual = actual->siguiente;
00124         }
00125     }
00126
00127     /**
00128     * @brief Muestra información de todos los sensores
00129     */
00130     void mostrarTodos() const {
00131         if (cantidad == 0) {
00132             std::cout << "\nNo hay sensores registrados" << std::endl;
00133             return;
00134         }
00135
00136         std::cout << "\n=== Sensores registrados: " << cantidad << " ===" << std::endl;
00137
00138         NodoSensor* actual = cabeza;
00139         while (actual != nullptr) {
00140             actual->sensor->imprimirInfo(); // Llamada polimórfica
00141             actual = actual->siguiente;
00142         }
00143     }
00144
00145     /**
00146     * @brief Obtiene la cantidad de sensores
00147     * @return Número de sensores
00148     */
00149     int getCantidad() const {
00150         return cantidad;
00151     }
00152 };
00153
00154 #endif

```

5.3. Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/ListaSensor.h

Lista enlazada genérica para almacenar lecturas de sensores.

```
#include <iostream>
```

Gráfico de dependencias incluidas en ListaSensor.h: Gráfico de los archivos que directa o indirectamente incluyen a este archivo:

Clases

- struct [Nodo< T >](#)
Nodo de lista enlazada genérico.
- class [ListaSensor< T >](#)
Lista enlazada genérica con gestión manual de memoria.

5.3.1. Descripción detallada

Lista enlazada genérica para almacenar lecturas de sensores.

Autor

Carlos Vargas

Fecha

30 de octubre de 2025

Definición en el archivo [ListaSensor.h](#).

5.4. ListaSensor.h

[Ir a la documentación de este archivo.](#)

```

00001 /**
00002  * @file ListaSensor.h
00003  * @brief Lista enlazada genérica para almacenar lecturas de sensores
00004  * @author Carlos Vargas
00005  * @date 30 de octubre de 2025
00006  */
00007
00008 #ifndef LISTA_SENSOR_H
00009 #define LISTA_SENSOR_H
00010
00011 #include <iostream>
00012
00013 /**
00014  * @struct Nodo
00015  * @brief Nodo de lista enlazada genérico
00016  * @tparam T Tipo de dato a almacenar
00017  */
00018 template <typename T>
00019 struct Nodo {
00020     T dato;          ///< Dato almacenado
00021     Nodo* siguiente; ///< Puntero al siguiente nodo
00022
00023     /**
00024      * @brief Constructor
00025      * @param valor Valor a almacenar
00026      */
00027     Nodo(T valor) : dato(valor), siguiente(nullptr) {}
00028 };
00029
00030 /**
00031  * @class ListaSensor
00032  * @brief Lista enlazada genérica con gestión manual de memoria
00033  * @tparam T Tipo de dato a almacenar (int, float, etc.)
00034  *
00035  * Implementa la Regla de Tres (constructor copia, operador=, destructor)
00036  * para gestión correcta de memoria dinámica.
00037  */
00038 template <typename T>
00039 class ListaSensor {
00040 private:
00041     Nodo<T>* cabeza; ///< Puntero al primer nodo
00042     int cantidad;    ///< Número de elementos
00043
00044     /**
00045      * @brief Libera toda la memoria de la lista
00046      */
00047     void liberar() {
00048         Nodo<T>* actual = cabeza;
00049         while (actual != nullptr) {
00050             Nodo<T>* temp = actual;
00051             actual = actual->siguiente;
00052             delete temp;
00053         }
00054         cabeza = nullptr;
00055         cantidad = 0;
00056     }
00057
00058     /**
00059      * @brief Copia profunda de otra lista
00060      * @param otra Lista a copiar
00061      */
00062     void copiar(const ListaSensor& otra) {
00063         if (otra.cabeza == nullptr) {
00064             cabeza = nullptr;
00065             cantidad = 0;
00066             return;
00067         }
00068
00069         cabeza = new Nodo<T>(otra.cabeza->dato);
00070         Nodo<T>* actualOtra = otra.cabeza->siguiente;
00071         Nodo<T>* actualEsta = cabeza;
00072
00073         while (actualOtra != nullptr) {
00074             actualEsta->siguiente = new Nodo<T>(actualOtra->dato);
00075             actualEsta = actualEsta->siguiente;
00076             actualOtra = actualOtra->siguiente;
00077         }
00078
00079         cantidad = otra.cantidad;
00080     }
00081
00082 public:

```

```

00083     /**
00084      * @brief Constructor por defecto
00085      */
00086     ListaSensor() : cabeza(nullptr), cantidad(0) {}
00087
00088     /**
00089      * @brief Destructor
00090      */
00091     ~ListaSensor() {
00092         liberar();
00093     }
00094
00095     /**
00096      * @brief Constructor de copia (Regla de Tres)
00097      * @param otra Lista a copiar
00098      */
00099     ListaSensor(const ListaSensor& otra) : cabeza(nullptr), cantidad(0) {
00100         copiar(otra);
00101     }
00102
00103     /**
00104      * @brief Operador de asignación (Regla de Tres)
00105      * @param otra Lista a asignar
00106      * @return Referencia a esta lista
00107      */
00108     ListaSensor& operator=(const ListaSensor& otra) {
00109         if (this != &otra) {
00110             liberar();
00111             copiar(otra);
00112         }
00113         return *this;
00114     }
00115
00116     /**
00117      * @brief Agrega un elemento al final
00118      * @param valor Valor a agregar
00119      */
00120     void agregar(T valor) {
00121         Nodo<T>* nuevo = new Nodo<T>(valor);
00122
00123         if (cabeza == nullptr) {
00124             cabeza = nuevo;
00125         } else {
00126             Nodo<T>* actual = cabeza;
00127             while (actual->siguiente != nullptr) {
00128                 actual = actual->siguiente;
00129             }
00130             actual->siguiente = nuevo;
00131         }
00132
00133         cantidad++;
00134     }
00135
00136     /**
00137      * @brief Calcula el promedio de los valores
00138      * @return Promedio (tipo T)
00139      */
00140     T calcularPromedio() const {
00141         if (cantidad == 0) return T(0);
00142
00143         T suma = T(0);
00144         Nodo<T>* actual = cabeza;
00145
00146         while (actual != nullptr) {
00147             suma = suma + actual->dato;
00148             actual = actual->siguiente;
00149         }
00150
00151         return suma / cantidad;
00152     }
00153
00154     /**
00155      * @brief Obtiene el número de elementos
00156      * @return Cantidad de elementos
00157      */
00158     int getCantidad() const {
00159         return cantidad;
00160     }
00161
00162     /**
00163      * @brief Imprime todos los valores
00164      */
00165     void imprimir() const {
00166         Nodo<T>* actual = cabeza;
00167         std::cout << "[";
00168         while (actual != nullptr) {
00169             std::cout << actual->dato;

```

```
00170         if (actual->siguiente != nullptr) {
00171             std::cout << ", ";
00172         }
00173         actual = actual->siguiente;
00174     }
00175     std::cout << "]\n";
00176 }
00177
00178 /**
00179  * @brief Verifica si la lista está vacía
00180  * @return true si está vacía
00181  */
00182 bool estaVacia() const {
00183     return cabeza == nullptr;
00184 }
00185 };
00186
00187 #endif
```

5.5. Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/SensorBase.h

Clase base abstracta para todos los sensores.

```
#include <iostream>
#include <cstring>
```

Gráfico de dependencias incluidas en SensorBase.h: Gráfico de los archivos que directa o indirectamente incluyen a este archivo:

Clases

- class [SensorBase](#)

Clase abstracta que define la interfaz para todos los sensores.

5.5.1. Descripción detallada

Clase base abstracta para todos los sensores.

Autor

Carlos Vargas

Fecha

30 de octubre de 2025

Definición en el archivo [SensorBase.h](#).

5.6. SensorBase.h

[Ir a la documentación de este archivo.](#)

```

00001 /**
00002  * @file SensorBase.h
00003  * @brief Clase base abstracta para todos los sensores
00004  * @author Carlos Vargas
00005  * @date 30 de octubre de 2025
00006  */
00007
00008 #ifndef SENSOR_BASE_H
00009 #define SENSOR_BASE_H
00010
00011 #include <iostream>
00012 #include <cstring>
00013
00014 /**
00015  * @class SensorBase
00016  * @brief Clase abstracta que define la interfaz para todos los sensores
00017  *
00018  * Esta clase utiliza polimorfismo para permitir procesamiento uniforme
00019  * de diferentes tipos de sensores a través de métodos virtuales puros.
00020  */
00021 class SensorBase {
00022 protected:
00023     char* id;           ///< Identificador único del sensor
00024     char* ubicacion;    ///< Ubicación física del sensor
00025
00026 public:
00027     /**
00028      * @brief Constructor parametrizado
00029      * @param id Identificador del sensor
00030      * @param ubi Ubicación del sensor
00031      */
00032     SensorBase(const char* id, const char* ubi) {
00033         this->id = new char[strlen(id) + 1];
00034         strcpy(this->id, id);
00035
00036         this->ubicacion = new char[strlen(ubi) + 1];
00037         strcpy(this->ubicacion, ubi);
00038     }
00039
00040     /**
00041      * @brief Destructor virtual (crítico para polimorfismo)
00042      */
00043     virtual ~SensorBase() {
00044         delete[] id;
00045         delete[] ubicacion;
00046     }
00047
00048     /**
00049      * @brief Procesa una lectura del sensor (método virtual puro)
00050      */
00051     virtual void procesarLectura() = 0;
00052
00053     /**
00054      * @brief Imprime información del sensor (método virtual puro)
00055      */
00056     virtual void imprimirInfo() const = 0;
00057
00058     /**
00059      * @brief Obtiene el ID del sensor
00060      * @return Puntero al identificador
00061      */
00062     const char* getId() const { return id; }
00063
00064     /**
00065      * @brief Obtiene la ubicación del sensor
00066      * @return Puntero a la ubicación
00067      */
00068     const char* getUbicacion() const { return ubicacion; }
00069 };
00070
00071 #endif

```

5.7. Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/SensorPresion.h

Sensor de presión atmosférica (valores enteros)

```
#include "SensorBase.h"
#include "ListaSensor.h"
```

Gráfico de dependencias incluidas en SensorPresion.h: Gráfico de los archivos que directa o indirectamente incluyen a este archivo:

Clases

- class [SensorPresion](#)
Sensor que mide presión atmosférica en hPa (tipo int)

5.7.1. Descripción detallada

Sensor de presión atmosférica (valores enteros)

Autor

Carlos Vargas

Fecha

30 de octubre de 2025

Definición en el archivo [SensorPresion.h](#).

5.8. SensorPresion.h

[Ir a la documentación de este archivo.](#)

```
00001 /**
00002  * @file SensorPresion.h
00003  * @brief Sensor de presión atmosférica (valores enteros)
00004  * @author Carlos Vargas
00005  * @date 30 de octubre de 2025
00006  */
00007
00008 #ifndef SENSOR_PRESION_H
00009 #define SENSOR_PRESION_H
00010
00011 #include "SensorBase.h"
00012 #include "ListaSensor.h"
00013
00014 /**
00015  * @class SensorPresion
00016  * @brief Sensor que mide presión atmosférica en hPa (tipo int)
00017  *
00018  * Hereda de SensorBase e implementa procesamiento específico
00019  * para datos de presión.
00020  */
00021 class SensorPresion : public SensorBase {
00022 private:
00023     ListaSensor<int> lecturas; ///< Lista de lecturas de presión
00024
00025 public:
00026     /**
00027      * @brief Constructor
00028      * @param id Identificador del sensor
00029      * @param ubi Ubicación del sensor
00030      */
00031     SensorPresion(const char* id, const char* ubi)
00032         : SensorBase(id, ubi) {}
00033
00034     /**
00035      * @brief Agrega una lectura de presión
00036      * @param valor Presión en hPa
00037      */
```



```

00038 void agregarLectura(int valor) {
00039     lecturas.agregar(valor);
00040 }
00041
00042 /**
00043  * @brief Procesa las lecturas de presión
00044  *
00045  * Calcula promedio y verifica límites (980-1050 hPa)
00046  */
00047 void procesarLectura() override {
00048     if (lecturas.getCantidad() == 0) {
00049         std::cout << " No hay lecturas" << std::endl;
00050         return;
00051     }
00052
00053     int promedio = lecturas.calcularPromedio();
00054     std::cout << " Promedio: " << promedio << " hPa" << std::endl;
00055
00056     if (promedio < 980) {
00057         std::cout << " ALERTA: Presion baja (tormenta)" << std::endl;
00058     } else if (promedio > 1050) {
00059         std::cout << " ALERTA: Presion alta" << std::endl;
00060     } else {
00061         std::cout << " Estado: Normal" << std::endl;
00062     }
00063 }
00064
00065 /**
00066  * @brief Imprime información del sensor
00067  */
00068 void imprimirInfo() const override {
00069     std::cout << "\n[PRESION]" << std::endl;
00070     std::cout << " ID: " << id << std::endl;
00071     std::cout << " Ubicacion: " << ubicacion << std::endl;
00072     std::cout << " Lecturas (" << lecturas.getCantidad() << "): ";
00073     lecturas.imprimir();
00074     std::cout << std::endl;
00075 }
00076
00077 /**
00078  * @brief Obtiene la lista de lecturas
00079  * @return Referencia a la lista
00080  */
00081 const ListaSensor<int>& getLecturas() const {
00082     return lecturas;
00083 }
00084 };
00085
00086 #endif

```

5.9. Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/SensorTemperatura.h

Sensor de temperatura (valores flotantes)

```
#include "SensorBase.h"
#include "ListaSensor.h"
```

Gráfico de dependencias incluidas en SensorTemperatura.h: Gráfico de los archivos que directa o indirectamente incluyen a este archivo:

Clases

- class [SensorTemperatura](#)
Sensor que mide temperatura en °C (tipo float)

5.9.1. Descripción detallada

Sensor de temperatura (valores flotantes)

Autor

Carlos Vargas

Fecha

30 de octubre de 2025

Definición en el archivo [SensorTemperatura.h](#).**5.10. SensorTemperatura.h**[Ir a la documentación de este archivo.](#)

```
00001 /**
00002  * @file SensorTemperatura.h
00003  * @brief Sensor de temperatura (valores flotantes)
00004  * @author Carlos Vargas
00005  * @date 30 de octubre de 2025
00006  */
00007
00008 #ifndef SENSOR_TEMPERATURA_H
00009 #define SENSOR_TEMPERATURA_H
00010
00011 #include "SensorBase.h"
00012 #include "ListaSensor.h"
00013
00014 /**
00015  * @class SensorTemperatura
00016  * @brief Sensor que mide temperatura en °C (tipo float)
00017  *
00018  * Hereda de SensorBase e implementa procesamiento específico
00019  * para datos de temperatura.
00020  */
00021 class SensorTemperatura : public SensorBase {
00022 private:
00023     ListaSensor<float> lecturas; ///< Lista de lecturas de temperatura
00024
00025 public:
00026     /**
00027      * @brief Constructor
00028      * @param id Identificador del sensor
00029      * @param ubi Ubicación del sensor
00030      */
00031     SensorTemperatura(const char* id, const char* ubi)
00032         : SensorBase(id, ubi) {}
00033
00034     /**
00035      * @brief Agrega una lectura de temperatura
00036      * @param valor Temperatura en °C
00037      */
00038     void agregarLectura(float valor) {
00039         lecturas.agregar(valor);
00040     }
00041
00042     /**
00043      * @brief Procesa las lecturas de temperatura
00044      *
00045      * Calcula promedio y verifica límites (15-30°C)
00046      */
00047     void procesarLectura() override {
00048         if (lecturas.getCantidad() == 0) {
00049             std::cout << " No hay lecturas" << std::endl;
00050             return;
00051         }
00052
00053         float promedio = lecturas.calcularPromedio();
00054         std::cout << " Promedio: " << promedio << " C" << std::endl;
00055
00056         if (promedio < 15.0f) {
00057             std::cout << " ALERTA: Temperatura baja" << std::endl;
00058         } else if (promedio > 30.0f) {
00059             std::cout << " ALERTA: Temperatura alta" << std::endl;
00060         } else {
00061             std::cout << " Estado: Normal" << std::endl;
00062         }
00063     }
00064 }
```

```
00065     /**
00066     * @brief Imprime información del sensor
00067     */
00068     void imprimirInfo() const override {
00069         std::cout << "\n[TEMPERATURA]" << std::endl;
00070         std::cout << " ID: " << id << std::endl;
00071         std::cout << " Ubicacion: " << ubicacion << std::endl;
00072         std::cout << " Lecturas (" << lecturas.getCantidad() << "): ";
00073         lecturas.imprimir();
00074         std::cout << std::endl;
00075     }
00076
00077     /**
00078     * @brief Obtiene la lista de lecturas
00079     * @return Referencia a la lista
00080     */
00081     const ListaSensor<float>& getLecturas() const {
00082         return lecturas;
00083     }
00084 };
00085
00086 #endif
```

5.11. Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/SensorVibracion.h

Sensor de vibración (valores enteros)

```
#include "SensorBase.h"
#include "ListaSensor.h"
```

Gráfico de dependencias incluidas en SensorVibracion.h: Gráfico de los archivos que directa o indirectamente incluyen a este archivo:

Clases

- class [SensorVibracion](#)
Sensor que mide intensidad de vibración (tipo int)

5.11.1. Descripción detallada

Sensor de vibración (valores enteros)

Autor

Carlos Vargas

Fecha

30 de octubre de 2025

Definición en el archivo [SensorVibracion.h](#).

5.12. SensorVibracion.h

[Ir a la documentación de este archivo.](#)

```

00001 /**
00002  * @file SensorVibracion.h
00003  * @brief Sensor de vibración (valores enteros)
00004  * @author Carlos Vargas
00005  * @date 30 de octubre de 2025
00006  */
00007
00008 #ifndef SENSOR_VIBRACION_H
00009 #define SENSOR_VIBRACION_H
00010
00011 #include "SensorBase.h"
00012 #include "ListaSensor.h"
00013
00014 /**
00015  * @class SensorVibracion
00016  * @brief Sensor que mide intensidad de vibración (tipo int)
00017  *
00018  * Hereda de SensorBase e implementa procesamiento específico
00019  * para datos de vibración/aceleración.
00020  */
00021 class SensorVibracion : public SensorBase {
00022 private:
00023     ListaSensor<int> lecturas; ///< Lista de lecturas de vibración
00024
00025 public:
00026     /**
00027      * @brief Constructor
00028      * @param id Identificador del sensor
00029      * @param ubi Ubicación del sensor
00030      */
00031     SensorVibracion(const char* id, const char* ubi)
00032         : SensorBase(id, ubi) {}
00033
00034     /**
00035      * @brief Agrega una lectura de vibración
00036      * @param valor Intensidad (0-100)
00037      */
00038     void agregarLectura(int valor) {
00039         lecturas.agregar(valor);
00040     }
00041
00042     /**
00043      * @brief Procesa las lecturas de vibración
00044      *
00045      * Calcula promedio y verifica niveles de alerta (0-100)
00046      */
00047     void procesarLectura() override {
00048         if (lecturas.getCantidad() == 0) {
00049             std::cout << " No hay lecturas" << std::endl;
00050             return;
00051         }
00052
00053         int promedio = lecturas.calcularPromedio();
00054         std::cout << " Promedio: " << promedio << std::endl;
00055
00056         if (promedio < 30) {
00057             std::cout << " Estado: Normal" << std::endl;
00058         } else if (promedio < 60) {
00059             std::cout << " ALERTA: Vibracion moderada" << std::endl;
00060         } else {
00061             std::cout << " ALERTA: Vibracion alta - revisar!" << std::endl;
00062         }
00063     }
00064
00065     /**
00066      * @brief Imprime información del sensor
00067      */
00068     void imprimirInfo() const override {
00069         std::cout << "\n[VIBRACION]" << std::endl;
00070         std::cout << " ID: " << id << std::endl;
00071         std::cout << " Ubicacion: " << ubicacion << std::endl;
00072         std::cout << " Lecturas (" << lecturas.getCantidad() << "): ";
00073         lecturas.imprimir();
00074         std::cout << std::endl;
00075     }
00076
00077     /**
00078      * @brief Obtiene la lista de lecturas
00079      * @return Referencia a la lista
00080      */
00081     const ListaSensor<int>& getLecturas() const {
00082         return lecturas;

```

```
00083     }  
00084 };  
00085  
00086 #endif
```

5.13. Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/include/SimuladorSerial.h

Simulador de comunicación serial Arduino.

```
#include <iostream>  
#include <cstdlib>  
#include <ctime>  
#include <cstring>
```

Gráfico de dependencias incluidas en SimuladorSerial.h: Gráfico de los archivos que directa o indirectamente incluyen a este archivo:

Clases

- class [SimuladorSerial](#)
Simula datos recibidos por puerto serial desde Arduino.

5.13.1. Descripción detallada

Simulador de comunicación serial Arduino.

Autor

Carlos Vargas

Fecha

30 de octubre de 2025

Definición en el archivo [SimuladorSerial.h](#).

5.14. SimuladorSerial.h

[Ir a la documentación de este archivo.](#)

```
00001 /**  
00002  * @file SimuladorSerial.h  
00003  * @brief Simulador de comunicación serial Arduino  
00004  * @author Carlos Vargas  
00005  * @date 30 de octubre de 2025  
00006  */  
00007  
00008 #ifndef SIMULADOR_SERIAL_H  
00009 #define SIMULADOR_SERIAL_H  
00010  
00011 #include <iostream>  
00012 #include <cstdlib>  
00013 #include <ctime>  
00014 #include <cstring>  
00015  
00016 /**  
00017  * @class SimuladorSerial
```

```

00018 * @brief Simula datos recibidos por puerto serial desde Arduino
00019 *
00020 * Genera datos aleatorios en formato "TIPO:ID:VALOR" simulando
00021 * el comportamiento del sketch Arduino.
00022 */
00023 class SimuladorSerial {
00024 private:
00025     bool inicializado; ///< Estado de inicialización
00026
00027 public:
00028     /**
00029      * @brief Constructor
00030      */
00031     SimuladorSerial() : inicializado(false) {}
00032
00033     /**
00034      * @brief Inicializa el generador de números aleatorios
00035      */
00036     void inicializar() {
00037         if (!inicializado) {
00038             srand(static_cast<unsigned int>(time(nullptr)));
00039             inicializado = true;
00040         }
00041     }
00042
00043     /**
00044      * @brief Genera una lectura de temperatura
00045      * @param buffer Buffer donde escribir el dato
00046      * @param tam Tamaño del buffer
00047      */
00048     void generarTemperatura(char* buffer, int tam) {
00049         float temp = 20.0f + (rand() % 100) / 10.0f;
00050         snprintf(buffer, tam, "TEMP:T-001:%.1f", temp);
00051     }
00052
00053     /**
00054      * @brief Genera una lectura de presión
00055      * @param buffer Buffer donde escribir el dato
00056      * @param tam Tamaño del buffer
00057      */
00058     void generarPresion(char* buffer, int tam) {
00059         int presion = 1000 + (rand() % 50);
00060         snprintf(buffer, tam, "PRES:P-105:%d", presion);
00061     }
00062
00063     /**
00064      * @brief Genera una lectura de vibración
00065      * @param buffer Buffer donde escribir el dato
00066      * @param tam Tamaño del buffer
00067      */
00068     void generarVibracion(char* buffer, int tam) {
00069         int vibracion = rand() % 80;
00070         snprintf(buffer, tam, "VIBR:V-201:%d", vibracion);
00071     }
00072
00073     /**
00074      * @brief Genera una lectura aleatoria de cualquier tipo
00075      * @param buffer Buffer donde escribir el dato
00076      * @param tam Tamaño del buffer
00077      */
00078     void generarLecturaAleatoria(char* buffer, int tam) {
00079         int tipo = rand() % 3;
00080
00081         switch (tipo) {
00082             case 0:
00083                 generarTemperatura(buffer, tam);
00084                 break;
00085             case 1:
00086                 generarPresion(buffer, tam);
00087                 break;
00088             case 2:
00089                 generarVibracion(buffer, tam);
00090                 break;
00091         }
00092     }
00093
00094     /**
00095      * @brief Simula recepción de datos del Arduino
00096      * @param buffer Buffer donde se escribirán los datos
00097      * @param tam Tamaño del buffer
00098      * @param numLecturas Número de lecturas a generar
00099      */
00100     void simularRecepcion(char* buffer, int tam, int numLecturas = 5) {
00101         inicializar();
00102
00103         std::cout << "\n=== Simulando Arduino ===" << std::endl;
00104         std::cout << "Generando " << numLecturas << " lecturas..." << std::endl;

```

```
00105
00106         for (int i = 0; i < numLecturas; i++) {
00107             generarLecturaAleatoria(buffer, tam);
00108             std::cout << " " << buffer << std::endl;
00109         }
00110     }
00111 };
00112
00113 #endif
```

5.15. Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/README.md

5.16. Referencia del archivo /home/charly/Documentos/Programacion Estructurada/sistema-sensores-iot/src/main.cpp

Programa principal SIMPLIFICADO del Sistema IoT.

```
#include <iostream>
#include <cstring>
#include <cstdlib>
#include "../include/ListaSensor.h"
#include "../include/SensorBase.h"
#include "../include/SensorTemperatura.h"
#include "../include/SensorPresion.h"
#include "../include/SensorVibracion.h"
#include "../include/ListaGestion.h"
#include "../include/SimuladorSerial.h"
```

Gráfico de dependencias incluidas en main.cpp:

Funciones

- void [mostrarMenu](#) ()
- void [procesarDatoArduino](#) (char *buffer, [ListaGestion](#) &listaGestion)
- int [main](#) ()

5.16.1. Descripción detallada

Programa principal SIMPLIFICADO del Sistema IoT.

Autor

Carlos Vargas

Fecha

30 de octubre de 2025

Definición en el archivo [main.cpp](#).

5.16.2. Documentación de funciones

main()

```
int main ()
```

Definición en la línea 90 del archivo [main.cpp](#).

Hace referencia a [SensorPresion::agregarLectura\(\)](#), [SensorTemperatura::agregarLectura\(\)](#), [SensorVibracion::agregarLectura\(\)](#), [ListaGestion::agregarSensor\(\)](#), [ListaGestion::buscarPorId\(\)](#), [SimuladorSerial::generarLecturaAleatoria\(\)](#), [SimuladorSerial::inicializar\(\)](#), [mostrarMenu\(\)](#), [ListaGestion::mostrarTodos\(\)](#), [procesarDatoArduino\(\)](#) y [ListaGestion::procesarTodosSensores\(\)](#).

Gráfico de llamadas de esta función:

mostrarMenu()

```
void mostrarMenu ()
```

Definición en la línea 21 del archivo [main.cpp](#).

Referenciado por [main\(\)](#).

Gráfico de llamadas a esta función:

procesarDatoArduino()

```
void procesarDatoArduino (  
    char * buffer,  
    ListaGestion & listaGestion)
```

Definición en la línea 34 del archivo [main.cpp](#).

Hace referencia a [SensorPresion::agregarLectura\(\)](#), [SensorTemperatura::agregarLectura\(\)](#), [SensorVibracion::agregarLectura\(\)](#), [ListaGestion::agregarSensor\(\)](#) y [ListaGestion::buscarPorId\(\)](#).

Referenciado por [main\(\)](#).

Gráfico de llamadas de esta función: Gráfico de llamadas a esta función:

5.17. main.cpp

[Ir a la documentación de este archivo.](#)

```

00001 /**
00002  * @file main.cpp
00003  * @brief Programa principal SIMPLIFICADO del Sistema IoT
00004  * @author Carlos Vargas
00005  * @date 30 de octubre de 2025
00006  */
00007
00008 #include <iostream>
00009 #include <cstring>
00010 #include <cstdlib>
00011 #include "../include/ListaSensor.h"
00012 #include "../include/SensorBase.h"
00013 #include "../include/SensorTemperatura.h"
00014 #include "../include/SensorPresion.h"
00015 #include "../include/SensorVibracion.h"
00016 #include "../include/ListaGestion.h"
00017 #include "../include/SimuladorSerial.h"
00018
00019 using namespace std;
00020
00021 void mostrarMenu() {
00022     cout << "\n=== Sistema IoT de Sensores ===" << endl;
00023     cout << "1. Crear Sensor Temperatura" << endl;
00024     cout << "2. Crear Sensor Presion" << endl;
00025     cout << "3. Crear Sensor Vibracion" << endl;
00026     cout << "4. Agregar Lectura Manual" << endl;
00027     cout << "5. Simular Arduino (5 lecturas)" << endl;
00028     cout << "6. Procesar Sensores" << endl;
00029     cout << "7. Mostrar Sensores" << endl;
00030     cout << "8. Salir" << endl;
00031     cout << "Opcion: ";
00032 }
00033
00034 void procesarDatoArduino(char* buffer, ListaGestion& listaGestion) {
00035     char tipo[10] = "";
00036     char id[20] = "";
00037     char valor[20] = "";
00038
00039     // Dividir el string "TIPO:ID:VALOR"
00040     char* token = strtok(buffer, ":");
00041     if (token) strcpy(tipo, token);
00042
00043     token = strtok(nullptr, ":");
00044     if (token) strcpy(id, token);
00045
00046     token = strtok(nullptr, ":");
00047     if (token) strcpy(valor, token);
00048
00049     cout << " Dato Arduino: " << tipo << " | " << id << " | " << valor << endl;
00050
00051     // Buscar sensor
00052     SensorBase* sensor = listaGestion.buscarPorId(id);
00053
00054     // Si no existe, crearlo
00055     if (sensor == nullptr) {
00056         if (strcmp(tipo, "TEMP") == 0) {
00057             sensor = new SensorTemperatura(id, "Arduino");
00058             listaGestion.agregarSensor(sensor);
00059         } else if (strcmp(tipo, "PRES") == 0) {
00060             sensor = new SensorPresion(id, "Arduino");
00061             listaGestion.agregarSensor(sensor);
00062         } else if (strcmp(tipo, "VIBR") == 0) {
00063             sensor = new SensorVibracion(id, "Arduino");
00064             listaGestion.agregarSensor(sensor);
00065         }
00066         sensor = listaGestion.buscarPorId(id);
00067     }
00068
00069     // Agregar lectura según tipo
00070     if (sensor) {
00071         SensorTemperatura* temp = dynamic_cast<SensorTemperatura*>(sensor);
00072         if (temp) {
00073             temp->agregarLectura(atoi(valor));
00074             return;
00075         }
00076
00077         SensorPresion* pres = dynamic_cast<SensorPresion*>(sensor);
00078         if (pres) {
00079             pres->agregarLectura(atoi(valor));
00080             return;
00081         }
00082     }

```

```

00083         SensorVibracion* vibr = dynamic_cast<SensorVibracion*>(sensor);
00084         if (vibr) {
00085             vibr->agregarLectura(atoi(valor));
00086         }
00087     }
00088 }
00089
00090 int main() {
00091     cout << "\n=== Sistema IoT - POO ===" << endl;
00092
00093     ListaGestion listaGestion;
00094     SimuladorSerial arduino;
00095     int opcion = 0;
00096
00097     do {
00098         mostrarMenu();
00099         cin >> opcion;
00100         cin.ignore();
00101
00102         switch (opcion) {
00103             case 1: {
00104                 char id[50], ubicacion[50];
00105                 cout << "ID del sensor (ej: T-001): ";
00106                 cin.getline(id, 50);
00107                 cout << "Ubicacion: ";
00108                 cin.getline(ubicacion, 50);
00109
00110                 SensorBase* sensor = new SensorTemperatura(id, ubicacion);
00111                 listaGestion.agregarSensor(sensor);
00112                 cout << "Sensor creado!\n";
00113                 break;
00114             }
00115
00116             case 2: {
00117                 char id[50], ubicacion[50];
00118                 cout << "ID del sensor (ej: P-105): ";
00119                 cin.getline(id, 50);
00120                 cout << "Ubicacion: ";
00121                 cin.getline(ubicacion, 50);
00122
00123                 SensorBase* sensor = new SensorPresion(id, ubicacion);
00124                 listaGestion.agregarSensor(sensor);
00125                 cout << "Sensor creado!\n";
00126                 break;
00127             }
00128
00129             case 3: {
00130                 char id[50], ubicacion[50];
00131                 cout << "ID del sensor (ej: V-201): ";
00132                 cin.getline(id, 50);
00133                 cout << "Ubicacion: ";
00134                 cin.getline(ubicacion, 50);
00135
00136                 SensorBase* sensor = new SensorVibracion(id, ubicacion);
00137                 listaGestion.agregarSensor(sensor);
00138                 cout << "Sensor creado!\n";
00139                 break;
00140             }
00141
00142             case 4: {
00143                 char id[50];
00144                 cout << "ID del sensor: ";
00145                 cin.getline(id, 50);
00146
00147                 SensorBase* sensor = listaGestion.buscarPorId(id);
00148                 if (sensor == nullptr) {
00149                     cout << "Sensor no encontrado!\n";
00150                     break;
00151                 }
00152
00153                 // Intentar convertir a cada tipo
00154                 SensorTemperatura* temp = dynamic_cast<SensorTemperatura*>(sensor);
00155                 if (temp) {
00156                     float valor;
00157                     cout << "Temperatura (C): ";
00158                     cin >> valor;
00159                     cin.ignore();
00160                     temp->agregarLectura(valor);
00161                     cout << "Lectura agregada!\n";
00162                     break;
00163                 }
00164
00165                 SensorPresion* pres = dynamic_cast<SensorPresion*>(sensor);
00166                 if (pres) {
00167                     int valor;
00168                     cout << "Presion (hPa): ";
00169                     cin >> valor;

```

```
00170         cin.ignore();
00171         pres->agregarLectura(valor);
00172         cout << "Lectura agregada!\n";
00173         break;
00174     }
00175
00176     SensorVibracion* vibr = dynamic_cast<SensorVibracion*>(sensor);
00177     if (vibr) {
00178         int valor;
00179         cout << "Vibracion (0-100): ";
00180         cin >> valor;
00181         cin.ignore();
00182         vibr->agregarLectura(valor);
00183         cout << "Lectura agregada!\n";
00184     }
00185     break;
00186 }
00187
00188 case 5: {
00189     arduino.inicializar();
00190
00191     cout << "\nSimulando 5 lecturas del Arduino...\n";
00192
00193     for (int i = 0; i < 5; i++) {
00194         char buffer[100];
00195         arduino.generarLecturaAleatoria(buffer, 100);
00196         procesarDatoArduino(buffer, listaGestion);
00197     }
00198     cout << "Simulacion completada!\n";
00199     break;
00200 }
00201
00202 case 6: {
00203     listaGestion.procesarTodosSensores();
00204     break;
00205 }
00206
00207 case 7: {
00208     listaGestion.mostrarTodos();
00209     break;
00210 }
00211
00212 case 8: {
00213     cout << "\nCerrando sistema...\n";
00214     break;
00215 }
00216
00217 default:
00218     cout << "Opcion invalida.\n";
00219 }
00220
00221 } while (opcion != 8);
00222
00223 cout << "\nSistema cerrado.\n";
00224 return 0;
00225 }
```

