TOMFramework Project

Class Design Document

Version: 0.1 (Draft version)

Authors:

|  |  |  |
| --- | --- | --- |
| No. | Student ID | Student name |
| 14 | B2110083 | Nguyen Phuc Nguyen Khoa |

*The purpose of this document is the draft version of class design that provides you with a guideline for detailing class design and writing the class diagram description for your solution based on design Pattern approach.*

*Points to remember:*

* *Content is important, not the volume.* ***Another team should be able to design in more detail from this document.***
* *Pay attention to overall in description.*
* *Completeness and consistency will be rewarded.*
* SINH VIÊN ĐƯỢC PHÉP XEM TÀI LIỆU
* SINH VIÊN KHÔNG ĐƯỢC SAO CHÉP HAY PHỔ BIẾN TÀI LIỆU TRONG PHÒNG THI DƯỚI MỌI HÌNH THỨC.

Contents

[1. Name: 4](#_Toc497804195)

[2. Problem 4](#_Toc497804196)

[3. Solution 4](#_Toc497804197)

[3.1. Class diagram 4](#_Toc497804198)

[3.2. Class Descriptions 4](#_Toc497804199)

[3.2.1. Class: <Classname> 4](#_Toc497804200)

[4. Consequence 5](#_Toc497804201)

## Name:

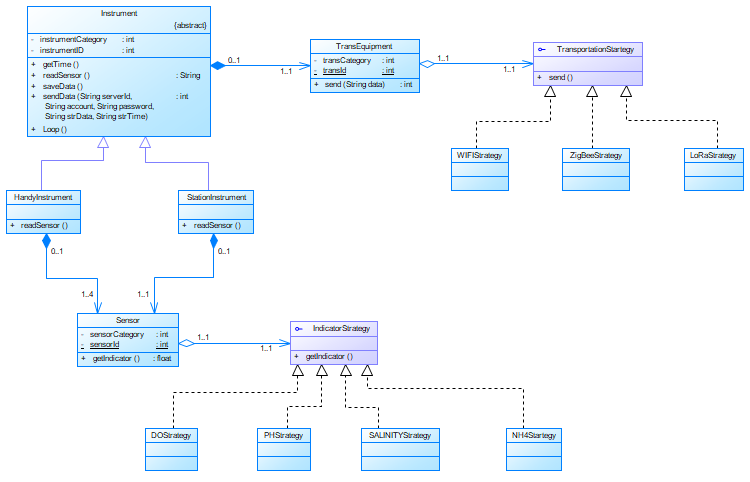
## Problem

*The project involves developing an embedded software solution for instruments that read environmental data using various sensors. The system must support up to 4 sensors (DO, PH, Salinity, NH4) and must be able to transmit the collected data via one of three communication technologies: WIFI, ZigBee, or LoRa. The challenge is to ensure that the system is flexible enough to accommodate various sensor types and transmission protocols, while also ensuring ease of integration and scalability for different use cases (handheld or station instruments).*

## Solution

*The solution involves designing a class structure using the Strategy pattern and Template method pattern to ensure modularity and flexibility. The Instrument class is abstract and can be extended by different instrument types like HandyInstrument and StationInstrument. Each sensor uses the IndicatorStrategy pattern, where specific sensor types like DO, PH, Salinity, and NH4 implement their own logic for reading data. For data transmission, we implemented a TransportationStrategy, where WIFI, ZigBee, and LoRa are different transmission strategies. This structure allows for easy extension and addition of new sensor types or communication protocols without significant changes to the core system.*

## Class diagram



## Class Descriptions

### Class: Instrument

* Purpose: *To model the relevant aspects of the Instrument*
* Constraints: *Each instrument must have at least one sensor and one transportation item*
* Persistent: *No (created at system initialization from other available data)*

Attribute Descriptions

1. *instrumentCategory:*

Type: int

Description: *Instrument category*

Constraints: *No null*

1. *instrumentId:*

Type: int

Description: *Instrument identifier*

Constraints: *Not null*

Method Descriptions

1. *getTime()*:

Return Type: *String*

Parameters: *None*

Return value: *A string that represents for DateTime in the format YYYY-MM-DD-hh:mm-ss*

Pre-condition: *The system clock is available and functioning*

Post-condition: *A formatted string representing the current date and time is returned*

Attributes read/used: *None*

Methods called: *Depends on system clock access*

Processing logic:

*- Access the system clock*

*- Retrieve the current date and time*

*- Format the date and time in the YYYY-MM-DD-hh:mm-ss format*

*- Return formatted string*

Test case:

*- Call getTime() at 16:00:00, October 10, 2024. Expected output is: “2024-10-10-16:00:00”*

1. *readSensor()*:

Return Type: *String*

Parameters: *None*

Return value: *A string in JSON format containing the readings from all attached sensors.*

Pre-condition: *At least one sensor is attached and operational*

Post-condition:Sensor values are retrieved, converted, and returned in a JSON string format.

Attributes read/used: *sensors*

Methods called: *Calls getIndicator() for each sensor attached.*

Processing logic:

*- Iterate over all sensors attached to the instrument.*

*- Call the getIndicator() method of each sensor to retrieve the sensor reading.*

*- Construct a JSON-formatted string containing the sensor data.*

*- Return the JSON string.*

Test case:

*- Test 1: Attach one sensor, call readSensor(), and expect a JSON string with the sensor's data.*

*- Test 2: Attach multiple sensors, call readSensor(), and expect a JSON string containing data from all sensors.*

1. *saveData()*:

Return Type: *void*

Parameters: *None*

Return value: *None*

Pre-condition: *A valid memory storage device is available and writable.*

Post-condition: *Sensor data is successfully written to the storage device.*

Attributes read/used: *Sensor data from readSensor().*

Methods called: *System-specific file handling methods.*

Processing logic:

*- Call readSensor() to obtain the sensor data*

*- Open the file on the external storage for writing*

*- Write the sensor data to the file*

*- Close the file after writing*

Test case:

*- Call saveData() and verify that the sensor data is stored in a file*

1. *sendData()*:

Return Type: *int (0 for success, non-zero for failure)*

Parameters:

*- String serverId: The server address to send data to*

*- String account: The account used for authentication.*

*- String password: The authentication password.*

*- String strData: The data to be transmitted (sensor data).*

*- String strTime: The timestamps of when the data was read.*

Return value: *Status code.*

Pre-condition: *A valid communication method (WIFI, ZigBee, LoRa) is available and configured.*

Post-condition: *Data is transmitted to the server, or an error code is returned.*

Attributes read/used: *Sensor data from readSensor(), timestamp from getTime().*

Methods called: *Calls send() on the appropriate TransEquipment strategy.*

Processing logic:

*- Retrieve sensor data using readSensor()*

*- Get the current timestamp using getTime().*

*- Pass the sensor data, timestamp, and connection details to the send() method of the active transmission device.*

*- Return the result of the send() method (0 for success, or error code).*

Test case:

*- Call sendData() with valid server credentials and expect a successful transmission (return 0).*

1. *Loop()*:

Return Type: *void*

Parameters: *None*

Return value: *None*

Pre-condition: *The system is powered on and all necessary components are initialized.*

Post-condition: *The system performs its tasks repetitively.*

Attributes read/used: *readSensor(), saveData(), sendData().*

Methods called: *Internal calls to getTime(), readSensor(), saveData(), sendData().*

Processing logic: *Continuously run the following operations:*

*- Get the current time.*

*- Read sensor values.*

*- Save the sensor data to memory.*

*- Send the sensor data to the server.*

Test case:

*- Run the loop and check if all functions (readSensor(), saveData(), sendData()) are called repeatedly.*

## Consequence

*TOMFramework provides a flexible approach for building embedded software for measuring environmental parameters, easily scalable and maintained when there are changes in the number or type of sensors and communication devices.*

## Implementation (code examples for Receive, HTTP and MQTT only)