CSE222 / BİL505
Data Structures and Algorithms
Homework #1
Due Date: 09/03/2025, 23:59

In this homework, you will design an interface for connecting multiple devices using different communication protocols. Using OOP techniques will help in reducing the complexity of interfacing modules using different communication protocols.

Scenario

You are asked to implement a framework for controlling communication ports of an embedded system. The system has multiple ports of distinct types (Protocols), each port can be occupied by a single device at a time. Devices can be sensors, displays, or wireless modules. The number of simultaneous devices of each type is limited. Each device type has its own methods for performing different operations. The user can add devices, remove devices, and execute operations on running devices. The user should also be able to view a list of connected devices and available ports when needed. Illegal operations such as adding devices beyond specified limits, attempting to connect multiple devices to the same port, or referencing a device using the wrong type should be detected and informed to the user WITHOUT TERMINATING THE PROGRAM.

Class Structure

Top System Class

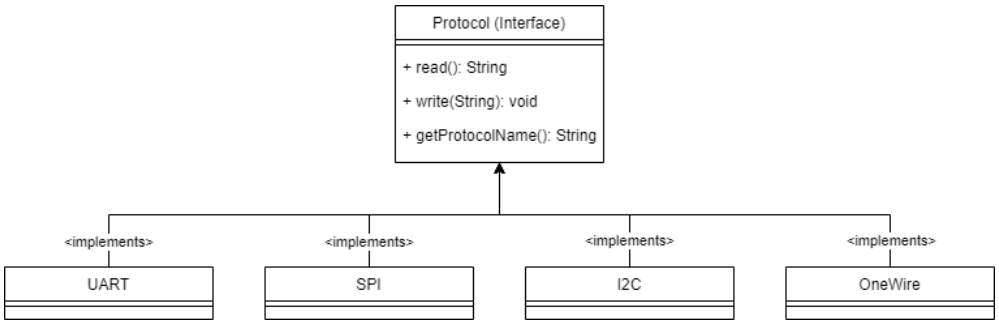
The HWSystem class will be the main interface for the user to do operations like adding devices, removing them, and viewing the state of the system. This class should provide the following operations:

- Add/remove devices.
- Turn devices ON/OFF.
- Do device specific operations:
    - Read sensor data.
    - Print data to display.
    - Change motor speed.
    - Send to/recieve from wireless adapter.
- Show a list of ports, showing both occupied and empty ports.
- Show lists of connected sensors/displays/motor drivers/wireless adapters.

Protocols

Protocols enable devices to communicate with the system without dealing with their lower-level complexity (Which is in this case a simple printf function). Here is the list of protocol classes in our system:

- Protocol (interface).
  - UART (implements Protocol).
  - SPI (implements Protocol).
  - I2C (implements Protocol).
  - OneWire (implements Protocol).



Protocol (Interface):

An interface that represents a protocol in the system. Declares the following methods:

- String getProtocolName(): Will return the name of the calling protocol. (“I2C”, “SPI”, etc.)
- String read(): Reads data from the port. Since this is artificial, it will return the following string: "<protocolName>: Reading.".
- void write(String data): Writes data to the port. Since this is artificial, it will print the following string to the terminal: "<protocolName>: Writing \“<data>\”.".

I2C, SPI, UART, and OneWire Classes:

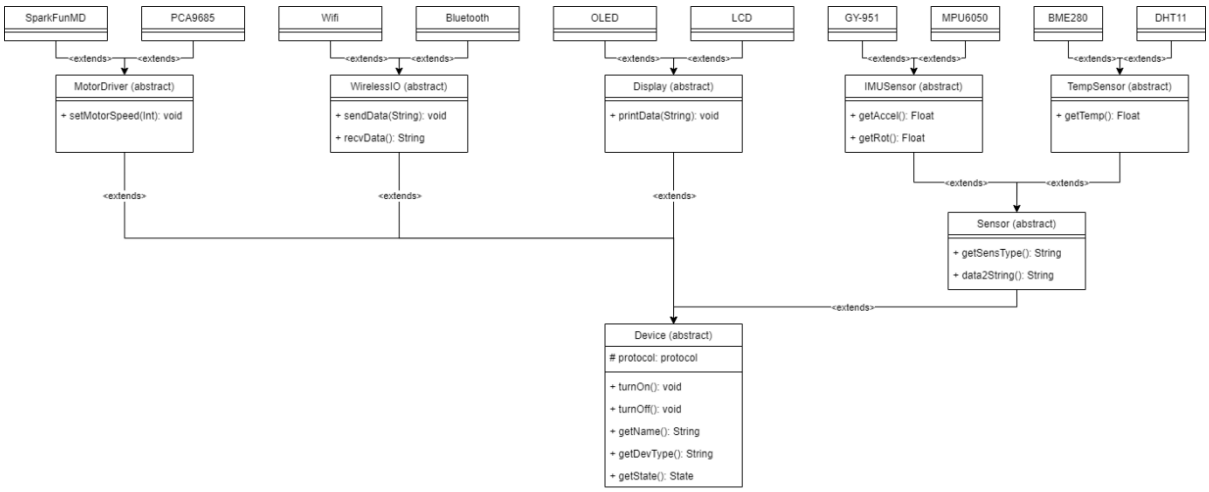
All these classes implement the Protocol interface. Since each of these classes implements the getProtocolName method differently, the feedback of read and write functions must be different.

Devices

Devices use protocols to communicate with the system that we are implementing. Devices have several types, and each type has multiple device models extending them. Each device type introduces a distinct set of operations. Here is the list of classes in the system:

- Device (abstract)
  - Sensor (extends Device) (devType)
    - TempSensor (Abstract) (extends Sensor)
      - DHT11 (extends TempSensor)
      - BME280 (extends TempSensor)
    - IMUSensor (Abstract) (extends Sensor)
      - MPU6050 (extends IMUSensor)
      - GY951 (extends IMUSensor)
  - Display (extends Device) (devType)

        - LCD (extends Display)
        - OLED (extends Display)
    - WirelessIO (extends Device) (devType)
        - Bluetooth (extends WirelessIO)
        - Wifi (extends WirelessIO)
    - MotorDriver (extends Device) (devType)
        - PCA9685 (extends MotorDriver)
        - SparkFunMD (extends MotorDriver)



Device (abstract):

An abstract class that represents a device connected to the system. Stores a reference to a Protocol object to be used during operations requiring communication. Declares the following methods:

- void turnON(): Changes the state of the device to ON. (Should call protocol write) (Should only be implemented by classes at the bottom of the hierarchy)
- void turnOFF(): Changes the state of the device to OFF. (Should call protocol write) (Should only be implemented by classes at the bottom of the hierarchy)
- String getName(): returns the name of the model of the device. (MPU6050, OLED, etc.) (Should only be implemented by classes at the bottom of the hierarchy)
- String getDevType(): returns the name of the type the device belongs to. (“WirelessIO”, “Display”, etc.) (Should be implemented by every device type, like Sensor and Display)
- State getState(): Return the state of the device. (State is an enum with ON/OFF values)

Sensor (abstract) (extends Device):

An abstract class that represents a sensor. Extends the Device interface and declares two new methods:

- String getSensType(): returns the name of the type the sensor belongs to. (“TempSensor”, “IMUSensor”)
- String data2String(): Convert the data read from the sensor to a string. Sensor types with multiple measurement functions should include all values.

Note: This class should implement getDevType method. This method should return a string in the following format: “<sensor type> Sensor”. <sensor type> is the output of getSensType().

TempSensor (abstract) (extends Sensor)

An abstract class that represents a generic temperature sensor. Introduces a new method, getTemp, that calls the read method from the assigned protocol and returns the temperature measured by the sensor as a Float (Random data is enough). Implements getSensType and data2String methods. data2String should return a string in the following format: "Temperature: %.2fC".

IMUSensor (abstract) (extends Sensor)

An abstract class that represents a generic IMU sensor. Introduces two new methods, getAccel and getRot, that call the read method from the assigned protocol and return the acceleration and rotation speed respectively as Floats (Random data is enough). Implements getSensType and data2String methods. data2String should call getAccel and getRot and return a string in the following format: "Accel: %.2f, Rot: %.2f".

Display (abstract) (extends Device)

An abstract class that represents a generic display. Introduces a new method, printData, that calls the write method from the assigned protocol with a given string parameter.

WirelessIO (abstract) (extends Device)

An abstract class that represents a generic wireless adapter. The class introduces two new methods, sendData and recvData, simulating send and receive operations of a wireless adapter. Both methods should use write and read methods of the assigned protocol, respectively.

MotorDriver (abstract) (extends Device)

An abstract class that represents a generic motor driver. Introduces a new method, setMotorSpeed, which calls the write method from the assigned protocol with the speed as the input parameter.

Device Classes

All the device classes at the bottom of the hierarchy should be concrete classes. Each class should have its unique name. These classes should implement device type-specific operations and turnON/turnOFF functions. Each device has specific protocol requirements, meaning that not all devices are compatible with all protocols. Here are the specifications:

- DHT11: OneWire.
- BME280: I2C, SPI.
- MPU6050: I2C.
- GY951: SPI, UART.
- LCD: I2C.
- OLED: SPI.
- Bluetooth: UART.
- Wifi: SPI, UART.
- PCA9685: I2C.
- SparkFunMD: SPI.

Configuration File

When launching the application, the information about port configuration and limits on device type numbers should be provided in a file in the following format: (line numbers are not included)

1. Port Configuration: <protocol>,<protocol>,<protocol>,.....
2. # of sensors:<integer>
3. # of displays:<integer>
4. # of wireless adapters:<integer>
5. # of motor drivers:<integer>

Runtime Commands

Commands will be given to the system through the terminal. Also, for test purposes, the input stream may be directed to a file to execute a large number of commands efficiently. Commands will have the following format:

- turnON/turnOFF <portID> (Calls Device.turnON/turnOFF method.)
- addDev <devName> <portID> <devID>
- rmDev <portID> (Should fail if the device is ON, or if the port is empty)
- list ports
  - output: (no information is needed for empty ports further than occupied/empty)
    list of ports
    <portID> <protocolType> <occupied/empty> <devName> <devType> <devID> <State>
    .
    .
- list <devType>
  - output:
    list of <devType>s
    <devName> <devID> <portID> <protocolType>
    .
    .
- readSensor <devID> (Calls Sensor.data2String method. Should fail if the device is OFF)
  - output:
    <devName> <devType>: <output of data2String>
- printDisplay <devID> <String> (Calls Display.printData method. Should fail if the device is OFF)
- readWireless <devID> (Calls WirelessIO.recvData method. Should fail if the device is OFF)
- writeWireless <devID> <String> (Calls WirelessIO.sendData method. Should fail if the device is OFF)
- setMotorSpeed <devID> <integer> (Calls MotorDriver.setMotorSpeed method. Should fail if the device is OFF)
- exit (exits the program)

Note: portID is the index of a port in the list of ports, they range from 0 to (max # of ports) - 1. devID is the index of a device among all devices of the same type, like sensors and displays, they range from 0 to (max # of devices of the same type) - 1.

Error Handling

The following illegal operations should be handled appropriately, and the user should be informed about them **WITHOUT TERMINATING THE PROGRAM**:

- Assigning a device to an incompatible or occupied port.
- Assigning a device with an already used or non-existing devID.
- Wrong input.
- Issuing a device type specific operation from a device that is OFF or a non-existing device.
- Removing a device that is ON.

Permissions and Restrictions:

- You cannot use any library other than the ones defined below:
  - java.io.File
  - java.util.ArrayList
  - java.util.Scanner (*not java.util.**)
  - Math library for random number generation.
- No downcasting.
- Access data fields only using methods.
- turnON/turnOFF operations should only be done on references of type “Device”.

General Information About the Homework:

- Cheating is not permitted. The students who cheat will receive NA from the course.
- In case of using AI tools for code generation, please specify it, otherwise, claiming ownership of code not your own will be penalized.
- You will be asked to attend to a demo to be graded. If you do not attend to the demo or cannot give proper answers to the questions you were asked during the demo, you will get a 0.
- The Problem Session on March 3 will describe the homework in detail. If you have any questions, you should ask them in PS. Any other questions before/after the PS will be ignored unless there is a mistake or missing information in this PDF. In such a case, the announcement will be made on Teams, you are responsible for reading them in time.
- Late submissions will not be allowed. The due date will not be postponed.

Runtime Example

Commands:

Command: list ports
list of ports:
0 I2C empty
1 SPI empty
2 OneWire empty
3 UART empty

Command: addDev DHT11 2 0

Command: list ports
list of ports:
0 I2C empty
1 SPI empty
2 OneWire occupied DHT11 TempSensor Sensor 0 OFF
3 UART empty

Command: list Sensor
list of Sensors:
DHT11 0 2 OneWire

Command: readSensor 0
Device is not active.
Command failed.

Command: turnON 2
DHT11: Turning ON
OneWire: Writing "turnON".

Command: readSensor 0
OneWire: Reading.
DHT11 TempSensor Sensor: Temp: 0.83

Command: rmDev 2
Device is active.
Command failed.

Command: turnOFF 2
DHT11: Turning OFF
OneWire: Writing "turnOFF".

Command: rmDev 2

Command: exit

Configuration File:

Port Configuration: I2C,SPI,OneWire,UART
# of sensors:1
# of displays:1
# of wireless adapters:1
# of motor drivers:1

Grading

Proper hierarchy structure	30 pts
Expected output for the specified operations	35 pts
Error Handling	35 pts
Total	100 pts

Penalties:

Code is not compilable	-100 pts (You get 0 directly)
Not attending demo / not providing acceptable explanations during demo	-100 pts (You get 0 directly)
Not a proper OOP design as described above	-100 pts (You get 0 directly)
Cheating	You get NA from the course
Too much reliance on AI	-50 pts
Additional library usage (other than the ones described above)	-30 pts
Missing operations	-10 pts (per operation)

PS: Additional grading criteria may apply.