ECE 448/528
Application Software Design

Lecture 9. HTTP Support for IoT Simulator
Spring 2025

Won-Jae Yi, Ph.D.

Department of Electrical and Computer Engineering
Illinois Institute of Technology

Object Oriented HTTP Server Design

Control IoT Simulator via Web Pages

- In our simple HTTP server design, `RequestProcessor` serves local files based on the path.
- One may dynamically generate content for HTTP response, without referring to any actual file.
  - e.g.) to create a report for the status of a smart plug for the path `/plugName`.
- One may additionally utilize the query to instruct the server to act differently.
  - e.g.) to switch a plug on by `/plugName?action=on`.
- Can we reuse `JHTTP` and `RequestProcessor`?
  - With minimal modifications.
  - Not just for our Project 2 but flexible enough for other purposes.

Polymorphism

- Allow `RequestProcessor` to react differently for different purposes.
- Possible method: Inheritance
  - Move code related to file handling in `RequestProcessor` to a new method.
  - New classes may inherit (`extends`) `RequestProcessor` and override this method to provide different services.
  - Update the `JHTTP` class as needed to create objects of such new classes instead of `RequestProcessor`.
- Limitations: no multiple inheritance in Java
  - Each Java class can only extend one superclass.
  - A choice made by Java: do not use inheritance (if possible).

Inversion of Control (IoC)

- An alternative method to support polymorphism in Java.
- Typical flow: an application uses a library by calling functions provided by the library.
  - e.g., socket programming.
- Inversion of Control (IoC): a library allows an application to provide code that the library will call.
  - *callbacks* in many other languages.
- Java `interface`
  - Help to define what the library expects from the application.
  - May contain one or more methods, each identifying one desired input/output behavior without any implementation.

Updated HTTP Server Design

- In the `ece448.iot_sim.http_server` package of the project code.
- `JHTTP`: allows it to run in its own thread.
  - We will have other things to do in the main thread so cannot allow `JHTTP` to wait for clients there.
- `RequestHandler`: a Java `interface` for HTTP request handling
  - The method `handleGet` corresponds to a GET request: the inputs are `path` and query `params` as key-value pairs, the output is an HTML page in `String`.
- `RequestProcessor`
  - Extract a single request. Parse path and query.
  - Let `RequestHandler` to generate a response.
  - Send the response back.
  - Generate error response when necessary.
- `RequestProcessor` will learn what `RequestHandler` to use from `JHTTP`.
  - `JHTTP` will depend on the application to provide it.

HTTP Support for IoT Simulator

Implementing RequestHandler

- Our IoT simulator needs to implement the `RequestHandler` interface for the HTTP server.
- `ece448.iot_sim.HTTPCommands`
  - Use a `TreeMap` to search for a particular plug.
  - List all plugs.
  - Report plug status.
  - Need code from you to switch plug on/off etc.
- How to write unit tests for it?

Concurrency and Synchronization

- `ece448.iot_sim.MeasurePower`: instruct plugs to measure their (hypothetical) power consumption every second.
- Requests to switch a plug on or off may come at the same time.
- A single `PlugSim` object may need to be accessed concurrently from multiple threads because of the above possibilities.
- **Synchronization**: an object shared by many threads needs to be protected by a *lock*.
  - So that at any moment, at most one thread can access it.
  - Each Java object has an intrinsic lock, and it can be activated via synchronized methods.
- All methods of `PlugSim` that read and write its state are `synchronized`.

Putting Everything Together

- Our simulator now begins to have many moving parts.
  - `PlugSim`'s for the plugs.
  - A `MeasurePower` object to control power measurements.
  - A `JHTTP` server to process HTTP requests.
  - A `HTTPCommands` object to handle HTTP requests for our application.
- `ece448.iot_sim.Main` takes care to initialize these objects properly before starting various threads.
  - Indeed, we will still follow the same steps when starting a more complex application.