GMU SWE 443 Software Architectures Spring 2012

Lab 1: Pipe-and-Filter

Sousa **Discuss:** Feb 2, **Due:** Feb 16

Problem Description

The objective of this assignment is to develop an appreciation of architectural patterns/styles and their impact on systemic properties. This assignment will use the pipe-and-filter architectural pattern as an exemplar. Part of this assignment will be implementation-oriented, allowing you to experiment with a particular pipe-and-filter implementation strategy in order to gain a clearer understanding of the issues associated with carrying an architectural design through to code. Note that this is not a programming class, and so the emphasis of this assignment is on the architecture issues – ***this cannot be stressed enough.*** Simply re-writing the entire implementation will indicate a lack of understanding of the core architectural concepts presented in class.

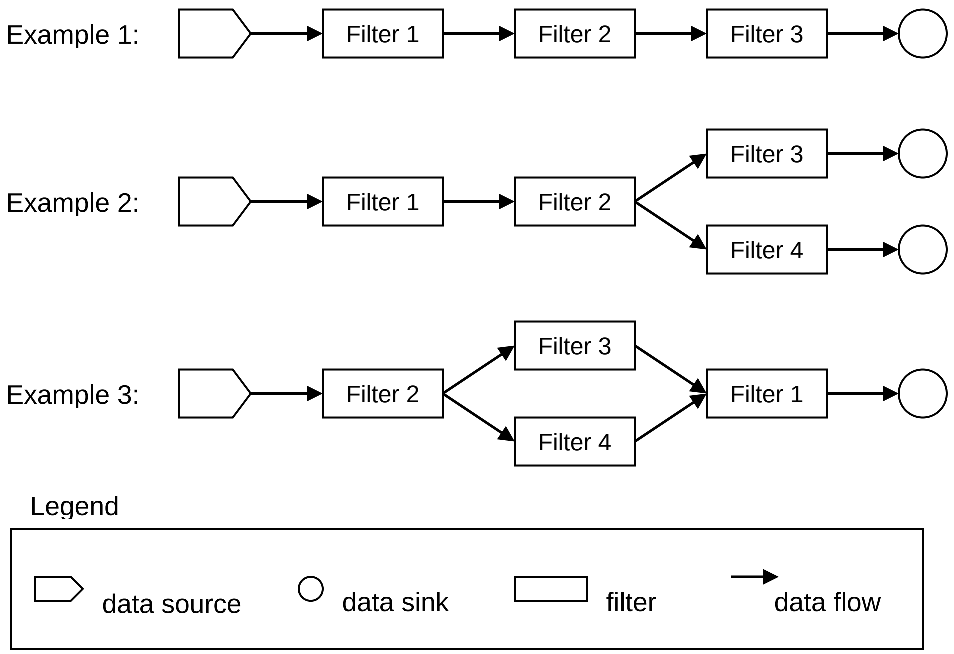
The assignment consists of two parts: For the first part of the assignment, you will be provided a working sample system that uses a (coding) framework supporting the pipe-and-filter paradigm. The application domain for this assignment is signal processing applications, as described below. Your task in part one is to extend the existing framework to architect and build the systems specified in the requirements below.

The second part of the assignment consists of analyzing the architecture of the system. After your analysis, design, and coding, you will reflect upon your work and answer questions related to the design decisions you made in part one. While you may discuss design decisions with your colleagues, the lab must be done individually.

Business Context and Key Architectural Approaches

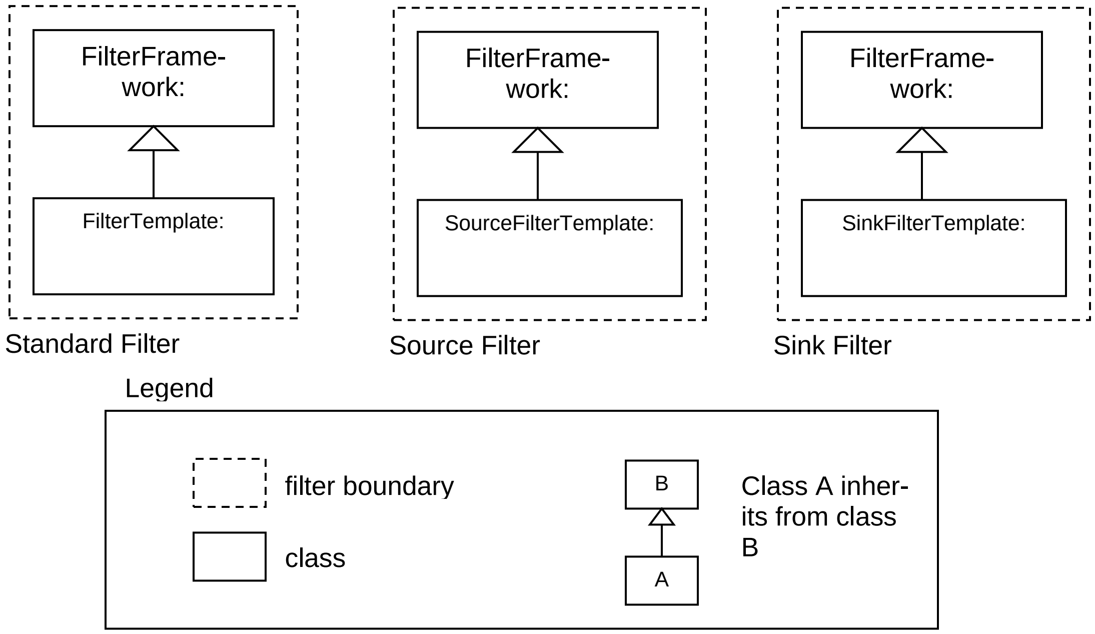
The principal stakeholder for this system is an organization that builds instrumentation systems. Instrumentation is a typical kind of signal processing application where streams of data are read, processed in a variety of ways, and displayed or stored for later use. A key part of modern instrumentation systems is the software that is used to process byte streams of data. The organization would like to create flexible software that can be reconfigured for a variety of applications and platforms (for our purposes, we can think of “platforms” as processors). For example, one application might be to support instrumentation for an automobile that would include data streams that originate with sensors and terminate in the cabin of the auto with a display of temperature, oil pressure, velocity, and so forth. Some subset of filters for this application might be used in aviation, space, or maritime applications. Another application might be in the lab reading streams of data from a file, processing the stream, and storing the data in a file. This would support the development and debugging of instrumentation systems. While it is critically important to support reconfiguration, the system must also process streams of data as quickly as possible. To meet these challenges, the architect has decided to design the system around a pipe-and-filter architectural pattern. From a dynamic perspective, systems would be structured as shown in the following examples.

Acknowledgment: D. Garlan & T. Lattanze @ CMU



The “data sources” in these systems are special filters that read data from sensors, files, or that generate data internally within the filter. All filter networks must start with a source. The “filters” shown in these examples are standard filters that read data from an upstream pipe, transform the data, and write data to a downstream pipe. The “data sinks” are special filters that read data from an upstream filter, but write data to a file or device of some kind. All filter networks must terminate with a sink that consumes the final data. Note that streams can be split and merged as shown in these examples.

The organization’s architect has developed a set of classes to facilitate the rapid development of filters and applications that can be quickly tested and deployed. These libraries have been provided to you. In addition there are several examples that have been provided to illustrate the use of these classes. The class structure (static perspective) for filters is as follows:



The FilterFramework class is the base class for all filters. It contains methods for managing the connections to pipes, writing and reading data to and from pipes, and setting up the filters as separate threads. Three filter “templates” have been established to ease the work of creating source, sink, and standard filters in a consistent way. Each of these filter templates describes how to write code for the three basic types of filters. Note that the current framework does not support splitting or merging the data stream. A fourth template, called the “PlumberTemplate” shows how pipe-and-filter *networks* can be set up from the filters created by developers. The “Plumber” is responsible for instantiating the filters and connecting them together. Once done with this, the plumber exits.

Data Stream format

The system’s data streams will follow a predetermined format of measurement ID and data point. Each measurement has a unique id beginning with zero. The ID of zero is always associated with time. Test files have been provided that contain test flight data that you will use for the project. The file data is in binary format – a data dump tool is provided to help read these files. The table below lists the measurements, IDs, and byte sizes of the data in these files.

ID	Data Descriptions and Units	Type	Number of Bytes
N/A	Measurement ID: Each measurement has an ID which indicates the type of measurement. The Measurement IDs are listed in this table in the left column.	Integer	4
000	Time: This is the number of milliseconds since the Epoch (00:00:00 GMT on January 1, 1970).	long Integer	8
001	Velocity: This is the airspeed of the vehicle. It is measured in knots per hour.	Double	8
002	Altitude: This is the vehicle’s distance from the surface of earth. It is measured in feet.	Double	8
003	Pressure: This is atmospheric pressure external to the vehicle. It is measured in PSI.	Double	8
004	Temperature: This is the temperature of the vehicle’s hull. It is measure in degrees Fahrenheit.	Double	8
005	Pitch: This is the angle of the nose of the vehicle relative to the surface of the earth. A pitch of 0 indicates that the vehicle is traveling level with respect to the earth. A positive value indicates that the vehicle is climbing; a negative value indicates that the vehicle is descending.	Double	8

Data in the stream is recorded in frames beginning with time, and followed by data with IDs between 1 and n, with n≤5. A set of time and data is called a frame. The time corresponds to when the data in the frame was recorded. This pattern is repeated until the end of stream is reached. Each frame is written in a stream as follows:

Frame 1	ID: 000	Time	ID: 001	Data	…	ID: n	Data
Frame 2	ID: 000	Time	ID: 001	Data	…	ID: n	Data
				:			
Frame F	ID: 000	Time	ID: 001	Data	…	ID: n	Data

Installing the Source Code

First, extract the files from “lab1.zip” file into a working directory. You will see four directories: *Templates*, *Sample*, *DataSets*, *HexDump*. The *Templates* directory contains the source code templates for the filters described above. The *DataSets* directory has all of the test data that you will need. The directory *HexDump* contains a utility that will allow you to read through the contents of the binary data files. The directory *Sample* contains a working pipe-and-filter network example that illustrates the basic framework. To compile the example in the *Sample* directory, open a command prompt window (or start a Linux command line terminal), change the working directory to *Sample*, and type the following:

```
...\assignment1\sample> javac *.java
```

The compile process above creates the class files. After you compile the system, you can execute it by typing the following:

```
...\assignment1\sample> java Plumber
```

Sample is a basic pipe-and-filter network that shows how to instantiate and connect filters, how to read data from the Flightdata.dat file, and how to extract measurements from the data stream.

The output of this example is measurement data and time stamps (when each measure was recorded) – all of this is written to the terminal. If you would like to capture this information to assist you in debugging the systems, you can redirect it to a file as follows:

```
...\assignment1\sample> java Plumber > output.txt
```

In this example, the output is redirected to the file output.txt.

Of course, to compile and run *Sample*, you may use Eclipse or any other IDE you prefer.

Part 1: Design and Construction

Your task is to use the existing framework as the basis for creating three new systems. Each new system has one or more requirements. In each system, please adhere to the pipe-and-filter architectural pattern as closely as possible. Make sure that you use good programming practices including comments that describe the role and function of any new modules, as well as describing how you changed the base system modules. Good programming practices will be appreciated.

System A

Create a pipe-and-filter network that will read the data stream in FlightData.dat file, convert the temperature measurements from Fahrenheit to Celsius, and convert altitude from feet to meters. Filter out the other measurements and write the output to a text file called OutputA.dat. Format the output as follows:

Time:	Temperature (C):	Altitude (m):
YYYY:DD:HH:MM:SS	TTT.ttttt	AAAAAA.aaaaa

System B

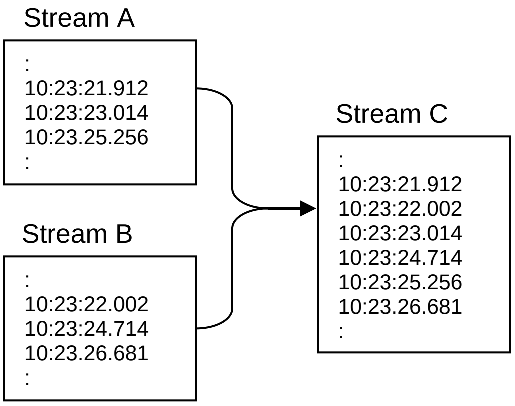
Create a pipe-and-filter network that does the same conversions as System A but keeps all fields. In addition, System B should filter “wild jumps” out of the data stream for altitude. A wild jump is a variation of more than 100m between two adjacent frames. For wild jumps encountered in the stream, interpolate a replacement value by computing the average of the last valid measurement and the next valid measurement in the stream. If a wild jump occurs at the end of the stream, replace it with the last valid value. Write the output to a text file called OutputB.dat and format the output as shown below, annotating any interpolated values with an asterisk:

Time:	Temperature (C):	Altitude (m):	Pressure (psi):
YYYY:DD:HH:MM:SS	TTT.ttttt	AAAAAA.aaaaa	PP:ppppp
YYYY:DD:HH:MM:SS	TTT.ttttt	AAAAAA.aaaaa*	PP:ppppp
YYYY:DD:HH:MM:SS	TTT.ttttt	AAAAAA.aaaaa	PP:ppppp
:	:	:	:

Write the records with rejected wild jumps to a second text file called WildPoints.dat using the same format as above.

System C

Create a pipe-and-filter network that merges two data streams. The system should take as input the SubSetA.dat file and the SubSetB.dat. Both of these files have the same 5 measurements as FlightData1.dat, which was recorded at different, but overlapping times. The system should merge these two streams together and time-align the data – that is, when the files are merged the single output stream’s time data should be monotonically increasing. This is illustrated below with a simple example. Here Stream C represents the merger of Stream A and Stream B.



In addition to merging the streams, you must filter the resulting data stream (Stream C) as follows:

1) Filter out pressure measurement wild points where the value exceeds 90 psi, or is less than 45 psi (yes this is intentionally different from System B). Write these values to a “rejected” file as required in System B. Replace any filtered wild points with interpolated values as you did in System B.

2) Filter out all measurements where: pitch exceeds 10 degrees and pressure exceeds 65 PSI and write them to another file using the same format used in System B. Replace any filtered wild points with interpolated values as you did in System B.

Packaging and submitting Part 1

- Systems A, B, and C should be clearly separated, both in terms of implementation and write-up (as described above). Place each implementation in a different folder for each of the systems.

- Part 1 must be emailed in a compressed folder named as LAB1-*yourLastName*.

- We will test your programs on our computers with test data sets that are similar to the data sets you have be provided with, but that may include more fields and or different data. You must clearly describe how to run your program. If you have any questions, please ask the instructor.

Part 2: System Analysis

Please answer the following questions. Each question has several parts. Make sure that you answer each question completely in your write-up:

1. For each system A, B, and C:
   - Describe the architecture of the system. Be sure to include appropriate views of the system. You are free to use whatever notations you prefer, but you must follow good standards of architectural documentation.
   - Are there other possible solutions that you could have adopted? What made you decide on your solutions over these other possible solutions?
2. Given your design and implementation of pipe-and-filter systems used for this assignment:
   - To what extent do your implementations (A, B, and C) differ from what is implied by an idealized notion of pipes and filters? Explain why and the impact it might have on systemic properties of your systems.
   - Is it possible for a developer to create circular dependencies in your systems? If so, what might be the result of executing a system with circular dependencies?

Grading Criteria

Your solutions and commentary will be graded based upon:

- The quality and contents of your write-up: this includes describing the design, discussions of trade-offs, and your discussion of the implications of changes to the system's architecture and the inherent systemic properties. Be clear, concise, and complete. This also includes the proper use of English prose, absence of grammar problems and misspellings.
- The consistency between the design representations and the implementation.
- The degree to which your solutions adhere to the pipe-and-filter architectural pattern where possible to do so.
- The extent to which deviations from the pipe-and-filter architectural pattern (as well as their effects) are clearly explained.
- Professionalism, which includes the quality of the report, timely submission, and well-structured and documented source code.
- The correct operation of the solutions - we will test your solutions with our test data.

Each assignment will be weighted as follows (100 points maximum):

Part 1: Implementation and Write-up

- Java implementation of System A: 20 points
- Java implementation of System B: 25 points
- Java Implementation of System C: 30 points

Part 2: Write-up

- Question 1: 15 points
- Question 2: 10 points