Group members:

Awis
Afin
Cnbl
Djam

Mini project 2

Flow:

The Source-Flush system is implemented in the 3 public classes, *AwesomeServer, Sink, and Source,* located respectively in .java files. In addition to the public classes, the AwesomeServer class contains an inner class by the name Client which serves as the abstraction of a subscribing connection which will be explained momentarily. Each of the public classes are intentionally to be run as independent applications through a shell of some as they actively record as well as print values through a console.

The structure of the system is as follows;
An AwesomeServer instance runs. Sources communicate with the AwesomeServer instance. Sinks inform the AwesomeServer instance of their existence, and receive/prints messages from them.

Optimal startup sequence: AwesomeServer > Sink > Source

AwesomeServer

This class has a ArrayList of inner-class Client objects which contains associated addresses and ports. This collection is used to keep track of the amount of sinks that needs to be communicated with. To register a sink, and effectively adding a sink-applications port and address, the server inspects all incoming messages, whereof a message containing the substring "**subscribe**" will do the job. All other messages, will be dispatched to the addresses found in the ArrayList (the sinks)

Source

This class merely implements an infinity loop to read input from the console. This input is then sent to the server, while discarding any response. To exit program, one must enter the string "exit".

Sink

This class initializes itself by sending a message containing the substring "**subscribe**" to effectively subscribe to messages from the server. After registering with the server, the instance starts listening on the same port of which it sended its subscribing message which is then receptacle for messages sent from the server that are printed to the console.

Questions:

1. Are your processes Web Services?

The definition of a Web service is a method of communicating two proccesses over a network in a specific XML format over typically HTTP. Since our Source is able to send messages but in string format over a local network using TCP to our Sink, we can say that our implementation is not directly a web service but can be used in the same way.

2. Is your system time and/or space de-coupled?

Because of the nature of the assignment, the system is space de-coupled but time coupled.

The messages are sent from the sources to the available sinks at a given time. Sinks which joins the system later on will not get previous messages and if the sink leaves the system it will not get any messages afterwards. Therefore a tight time coupling exists in the system.

The sources and sinks know nothing about each other, so any number of sources can send messages to any number of sinks. Therefore the system is space de-coupled.s

3. Is your system a message queue?

The system does not keep messages queued, but tries sends them from source to sink at the instant. Therefore the implementation is limited by the amount of messages it can send and recieve at a given time. If the socket is busy the message will be dropped.

4. Is your system a publish/subscribe system?

Yes the system works like a Publish/Subscribe system. Sinks automatically subscribes to all the active sources (including sources which joins in later). Sources publish their content and when they do, all subscribers(sinks) gets notified by receiving the published content.

5. What are the failure modes of your system?

a) A single instance of AwesomeServer exists at all times. Attempts to create multiple instances will result in a startup exception as the port will be occupied by the initial instance.
b) Sinks may at times be blocked in the case that a high amount of messages are dispatched from the server. This raises an exception on the server instance as it is unable to connect to the sink which is processing a previous message. This could quite easily be fixed by delegating the client object returned to a separate thread - this effectively unblocks the listening port.