Coding Challenge

Please pick one from the following challenge descriptions.  You have as long you’d like to complete the challenge.  Treat it as you would if you were writing code for Mobiquity in your day to day job - after all this is the only way we can evaluate what your coding style and standards are.

Please implement the challenge in Node.js and have it available for viewing in GitHub.

Comments are good.  Clear, concise code is better.  And btw, we are fanatical about tests.

Challenge 1 - Implement a simple pub/sub

In this challenge, you will implement a simple in-memory publish/subscribe bus interface.

Interested parties may “subscribe” to “topics”.  Publishers may “publish” a message to topics.

Each listener should be called back with the message if a message is published to a topic for which it has previously subscribed.

**Requirements**

* Implement a class with the methods (interface definition is deliberately agnostic, you define the language and the actual definition):
  + void subscribe(topic, listener)
  + void publish(topic, message)
* When a listener is subscribed to a topic, it should be called back with a message that is published to that topic.
* Ensure the class is thread safe (if using a language that supports threading, e.g. Java)

**Extra Credit**

* Implement wildcard specifiers, e.g. subscribe("f\*", listener)
* Implement queueing and history meaning that a history of messages delivered to the topic are recorded and the most recent two messages should be delivered upon subscription to the topic - e.g. pubsub.subscribe("foo", listener2, 2)

**Sample code (pseudo-code)**

listener1.callback = function(msg) {

   println('listener1: ' + msg);

}

listener2.callback = function(msg) {

   println(‘listener2: ‘ + msg);

}

pubsub.subscribe("foo", listener1);

pubsub.subscribe("foo", listener2);

pubsub.subscribe("bar", listener2);

pubsub.publish("foo", "hello");

pubsub.publish("bar", "world");

**Sample output**

listener1: hello

listener2: hello

listener2: world

Challenge 2 - Display a graph

In this challenge, write a simple program that can read a dot file.  An explanation of the dot language is here <http://en.wikipedia.org/wiki/DOT_(graph_description_language)>.  For this exercise you need only implement a simple rendering engine that supports unidirectional dependencies.  Entities need not be rendered in any special way other than their name.

**Requirements**

* Provide a facility to read and display a dot file.  Reading from a file is acceptable. So is reading from stdin.
* Display a dependency list from the given file.  Plain text formatting is acceptable.  Writing to stdout is acceptable.  So is generating a web page.  Your choice
* You do not need to concern yourself with complex features of the language

**Extra Credit**

* Support advanced formatting features of the language

**Sample Input**

digraph graphname {

    a -> b -> c;

    b -> d;

}

**Sample output**

a

+--> b

    +--> c

    +--> d