Concurrent Programming in Erlang

In this exercise we'll look at a few ways in which the frequency server can be enhanced. You should submit a revised version of the `frequency.erl` file, with appropriate comments added to explain what you have done.

You'll get a chance to to give other learners feedback on their work, as well as getting feedback on your own. This exercise is also reproduced as a PDF file and with a supporting file `frequency2.erl`, both available on the previous step, as once you've submitted the exercise instructions will no longer be available here.

The guidelines below indicate what we would like you to do do in your own work and look for in others'. To include a comment in your Erlang code use the `%` symbol: this makes the remainder of the line into a comment, as in

```
%%this is a comment
-module(ex).
f(0) -> 0;
f(N) -> N + f(N-1). % non-zero
case
```

Be aware that formatting functions are limited in this platform. You may prefer to use the assignment submission box just to present commentary on your submission (for example why you chose a particular approach, or any difficulties you found) and include within that submission a **link** to your actual assignment, using a service of your choice - such as a github gist, hastebin file, Dropbox or Google Drive.

Please note: you will not be able to edit your work after clicking 'Submit', so please check it carefully before submitting.

Now that we have introduced the functional interface to the frequency server client code, it is possible to enhance its functionality without changing the interface.

Try making the following modifications, behind the functional interface. Submit a revised version of the `frequency.erl` file, with appropriate comments added to explain what you have done.

Flushing the mailbox

Suppose that we want to ensure that any messages that happen to be in a mailbox are removed. We might think that we could remove them all like this:

```
clear() ->
  receive
    _Msg ->
clear()
  end.
```

But this has two problems. First, it will block if no messages are present, and, second, it will never terminate. The way to ensure that it only processes messages that are already in the mailbox, and terminates once they are removed, is to use a timeout of zero. Modify the definition of `clear/0` to include this.

Adding timeouts to the client code

Suppose that the frequency server is heavily loaded. In this case it could make sense to add timeouts to the client code that asks to allocate or deallocate a frequency. Add these to the code.

One possibility when a `receive` times out is that a message is subsequently delivered into the mailbox of the receiving process, but not processed as it should be. It can then become necessary to clear the mailbox periodically: where would you add these calls to `clear/0`?

You can simulate the frequency server being overloaded by adding calls to `timer:sleep/0` to the frequency server. If these delays are larger than the timeouts chosen for the client code, you will be able to observe the late delivery of messages by modifying `clear/0` to print messages as they are cleared from the mailbox. Define a modified version of `clear/0` to do this, and test it with your “overloaded” server.
