1. Ok, one more time! Implement the pi-calculation example using Erlang actors. Your module should export a function `doRun(NumIterations, NumActors)` that returns the approximated value of pi, using the given number of iterations per actor and the given number of actors. (Note, since actors only communicate by message passing, each actor can return its result to the main process via a message; the main process can receive messages and tally up the results, returning the result after `NumActors` messages have been received.)

2. Modify the code in `hw6.zip` to use ICE for the client and server. (hw6.zip is essentially the same code we saw before in the package examples/week5/callbacks_and_futures with redundant stuff removed.) The intent of this exercise is to see how easy it is to use a framework such as ICE to set up a basic asynchronous, distributed system (particularly if you are using Java).

Before starting this problem, be sure you can build and run the simple Hello, World example. See the section "Notes on ICE" below to get started.

Start with a new, empty Eclipse project (or the equivalent, if you are not using Eclipse). Use the following slice definition,

```plaintext
module dbexample {

    exception NoSuchEntryException { }

    interface Database {
      string retrieve(int key) throws NoSuchEntryException;
    }
}
```

and create the package `dbexample` in your Eclipse `src` directory before running slice2java, as in the example below under "Notes on ICE". Create two new packages, say `client` and `server`.

Copy `ClientWithCallback` into the client package and modify it to use the generated interface `DatabasePrx`. Modify the callback class to extend the generated abstract class `Callback_Database_retrieve` and implement the abstract methods appropriately. In the main method, put the code that starts up the communicator more or less as in the hello world example. However, instead of exiting right away as in that example, just enter into the client's `go()` loop, which will keep the communicator alive. You'll need to modify the constructor of `ClientWithCallback` so you can pass in the `DatabasePrx` object. Modify the imports so it uses the generated type `dbexample.NoSuchEntryException`.

Copy `FakeDatabase` into the server package and add a "servant" class `DatabaseI` that extends the generated class `_DatabaseDisp`, just as for the hello world example. Copy over the `Server` class from the hello world example and make the minor modifications needed to create the `DatabaseI` servant instead of the printer servant.
Finally, you'll notice that your requests are sent asynchronously, but by default the server is single threaded (try sending a whole bunch of keys in succession). Configure the thread pools on client and server, respectively, by passing command line arguments to the `initialize` method, e.g.,

```java
```

for the client and

```java
```

for the server.

**Notes on ICE**

Download and install ICE for your platform from [https://zeroc.com/distributions/ice/3.5/](https://zeroc.com/distributions/ice/3.5/) *(Note that this is not the latest version of ICE. The current version, 3.6, introduces some extra dependencies (at least on OS X) that are not worth the extra trouble. This may not be an issue on Windows.)*

Make sure you can run the tools from a command shell, e.g. try executing the command `slice2java`. You may need to put the ICE `bin` directory in your path.

The code for the Hello, World example can be directly copied and pasted from the ICE manual at

[https://doc.zeroc.com/display/Ice35/Hello+World+Application](https://doc.zeroc.com/display/Ice35/Hello+World+Application)

(See "Writing a Slice Definition" and "Writing an Ice Application with Java")

Here is a quick summary of the steps. Note in the steps below we pre-create a package for the generated code within Eclipse. Then everything can be compiled via Eclipse rather than worrying about compiling from the command line as described in the ICE documentation.

1. Create a new Eclipse project, say `ice_hello`.
2. Make the file `Printer.ice` described in the ICE manual and save it in the `ice_hello` project directory.
3. Create a package called Demo.
4. In command shell, cd to the `ice_hello` project directory
5. Run the command `slice2java --output-dir src Printer.ice` (this will put the generated files in the package directory Demo under `src`)
6. In Eclipse, right-click on Demo package and select Refresh; you should see all the generated files, and lots of compile errors
7. Add Ice.jar to the build path. It is located in the ICE lib directory. Import it into the project, right-click, and select Build Path -> Add to Build Path. Hopefully no compile errors now.
8. Create the `PrinterI "servant" class to extend Demo._PrinterDisp`, as described in the documentation. *(Don't put it in the Demo package; here you can put this in the default package.)*
9. Create **Server** and **Client** main classes, copy and pasting code from the ICE documentation. It is pretty much boilerplate, but the documentation explains what it's doing.

10. Run the **Server**, then run the **Client**. Check the server's console pane for the "Hello, World" message. It is easier to see what's going on if you run the server in a separate command shell; just cd to the project directory and run `java -cp bin:Ice.jar Server` (if you are on Linux or OS X use a colon instead of a semicolon, of course).