COMP 4958: Lab 3

Submit a zip file named `lab3.zip` containing the file `arithmetic.ex` (for part 2) and the `card` folder with its files, except for the `_build` directory (for part 1). Maximum score: 13

1. Re-implement the "card server" from lab 2 as a GenServer. Use a `mix` project named `card` and put the code in a module named `Card.Server`. Unlike the server from lab 2, this new version is unregistered, which means that some of its API functions need to take a PID as an argument. Client API:

   ```
   start()       # start unregistered server with "sorted" deck of 52 cards
   new(pid)      # use a new "sorted" deck of 52 cards
   shuffle(pid)  # shuffle deck of (remaining) cards
   count(pid) => integer  # return number of remaining cards
   deal(pid, n \\ 1) => {:ok, [card]} | {:error, reason}  # ask server to deal n cards
   ```

   Note that we can run multiple copies of this server as it is unregistered.

2. Execution within a single process is sequential. A server process that performs lengthy operations may not be able to handle a large volume of requests within a short period. One solution is to have a server that creates a pool of worker processes; the server dispatches requests to the workers. This works particularly well if the workers do not need to store state.

   There are different ways of dispatching requests to the workers. One way is round-robin: The first request goes to worker 1, the next to worker 2, and so on, until we get to the last worker. Then we wrap around back to worker 1.

   For this exercise, we'll use the simple "arithmetic worker" from class. Modify the `square` and `sqrt` function so that the worker returns its PID together with the result of the calculation. (This makes it easy to see which worker is handling a particular request.) To simulate a lengthy calculation, have the worker sleep for 4 seconds when handling `sqrt` requests. Put your implementation in a module named `Arithmetic.Worker`.

   In a separate module named `Arithmetic.Server`, implement the server, which is registered. Its `start` function takes a positive integer that specifies the number of workers to start. The worker's PIDs are displayed when they start. This server provides the same operations as the workers: `square(x)` and `sqrt(x)`. These only take the number as an argument; the PID is not required as the server is registered. The server chooses a worker (in round-robin fashion) to handle the (square or square root) request. The server does not wait for a worker to finish the calculation.

   Put both `Arithmetic.Worker` and `Arithmetic.Server` in a file named `arithmetic.ex`. Both are GenServers.