1 The Challenges of Concurrency

1.1 Mysterious Money

1. Suppose that Andrew and Rohin decide to pool some money together:

```python
>>> balance = 100
```

Now suppose Rohin deposits $20, and Andrew withdraws half of the money in the account by executing the following commands:

- Rohin: `balance = balance + 20`
- Andrew: `balance = (balance / 2)`

Suppose this “did the right thing” (we’ll formalize this later). What would be the possible values of `balance` after all the commands have finished executing?

2. What are the values that could be produced if the system allows the processes to be interleaved?
1.2 Correctness

From the previous question, it should hopefully be intuitively clear that when you have two or more programs running in parallel, the output of that is considered to be “correct” if you could get the same output by running the programs sequentially in some order. Note that it is not necessary for the programs to actually be run sequentially - this would defeat the entire purpose of concurrency - it only needs to behave as though it were run sequentially.

However, in many cases if we take two programs and run them concurrently, we could get incorrect results. The challenge is then how to design programs that can run concurrently but still always give correct results.

2 Locks

Locks are shared objects that are used to signal that shared state is being read or modified. They are also referred to as a mutexes (short for "mutual exclusion"). A lock has the acquire and release methods. Once a lock is acquired, no other program can acquire the lock until the original program that acquired the lock releases the lock. In this way, locks allow us to enforce that certain parts of the code must execute sequentially.

1. What are the possible values of executing the following two programs concurrently?
   Assume that lock is a lock that both programs have access to, and x and y are variables that both programs have access to that initially have values x = 3 and y = 5.

   ```
   lock.acquire()
   x = x + y
   lock.release()
   y = y + 1
   lock.acquire()
   y = y + 2
   lock.release()
   ```

2.1 Locks in Java

In Java, every object has a built-in lock. You do not need to access this lock directly - you can simply use the synchronized keyword in order to automatically use the lock. Use Google to find out more!