Homework #4: Go

Issued: Tuesday, November 7
Due: Thursday, November 16

Purpose

This assignment allows you to program in a language introduced in lecture: Go.

Why Go?

Go is a garbage-collected, concurrent, object-oriented, and imperative language. It has been considered a competitor of the systems-programming duo of C and C++, but it has garbage collection. Rust is also a competitor in this arena, without garbage collection.

Go was designed by Robert Griesemer, Rob Pike, and Ken Thompson, at Google, in 2009.

Submission

Homework is due at 11:59PM, Mountain Time, on the day it is due. Late work is not accepted. To submit your solution to an assignment, login to a lab computer, change to the directory containing the files you want to submit, and execute:

```
submit jbuffenb class assignment
```

For example:

```
submit jbuffenb cs354 hw1
```

The `submit` program has a nice `man` page.

When you submit a program, include: the source code, sample input data, and its corresponding results.

Scores are posted in our `pub/scores` directory, as they become available. You will receive a code, by email, indicating your row in the score sheet. You are encouraged to check your scores to ensure they are recorded properly. If you feel that a grading mistake has been made, contact me as soon as possible.

Documentation

Go lecture slides are at:

```
pub/slides/slides-go.pdf
```

Go is demonstrated by:

```
pub/sum/go
pub/etc/sleepsort
```

Go is too new to be described in our textbook.

Links to programming-language documentation can be found at:

```
http://csweb.boisestate.edu/~buff/pl.html
```

Assignment

Begin, by porting the simple banking application at:

```
pub/hw4
```

from Java to Go. Model your Go solution on the Java solution. Thus, you will have multiple Go packages. Use the patterns we saw in:

```
pub/etc/student
```

Then, reimplement the `Accrue` functions, to use goroutines. This is a bit silly, since interest-accrual is too simple to deserve separate threads. In any event,

use a channel, so the bank's `Accrue` function can sum the interest, and print the total. Use the patterns we saw in:

```
pub/etc/sleepsort
```

Hints and Advice

- An abstract class/method can be approximated with an interface that declares a function that has no definition.
- Your bank can represent its set of accounts as a map from interface pointers to interface values. An interface value is essentially an object reference:

  ```
  accounts map[*IAccount]IAccount
  ```

  and iterate over them like this:

  ```
  for _,a:=range b.accounts {
      ...
  }
  ```