GO Concurrency – Grocery Store

Class: CS315 – Distributed Scalable Computing
Whitworth University, Instructor: Scott Griffith

Last Modified: 10/4/2018

Part 1: Concurrent Store (100pts)

Remember the first week of class when we talked about the concurrent operations of a **Grocery Store?** This week we are going to make a simple simulation of a store using multiple agents and Go channels to manage data movement.

You are going to make a number of different agents that will all do their work concurrently. These will be implemented with **multiple goroutines**.

To call a function **asynchronously**:

```
go myfunction(arguments)
```

Your store will be a store that must sell two things: Pepsi and Coke. You as the owner and operator are allowed to sell other things, but are required to stock Pepsi and Coke.

For your store to work, you are going to need to implement the following **agents** (asynchronous functions):

- Coke Delivery Person
- Pepsi Delivery Person
- Shelf Stocking Person
- A number of Customers
- A Check-out Person

You are also going to need a **shelf to hold all your stock**. (that is a resource, not an agent!)

Your **delivery people** will deliver product through the day in amounts of 24 cans (1 case). In order to make sure they get paid, they will also record a **running bill** that the store will have to reconcile at the end of the day. Each can of coke they deliver is going to cost the store $.25, each can of Pepsi will be $.20. Each delivery person should deliver a total of 480 cans of soda per day, but never more often than every .5 seconds.

The delivery people will give their product to the **stocker** who will then stock the shelf. The stocker is really lazy and can only place one can on the shelf every 10 milliseconds. The stocker has no preference on brand, he will take from either the coke or the pepsi delivery as he chooses.

A **variable number of customers** will come in each day and buy **varying amounts of soda**. If the shelf is empty, they will wait around until the end of the day before leaving. Once they pick up their soda, they will proceed to the Check-out Person to pay for their product. These customers have unlimited supply of money, they can always pay. Each person will buy **1 to 4 six packs** (so 6,12,18 or 24 cans of soda). There will be at least 30 customers buying Pepsi and at least 30 customers buying Coke through the day. The store only allows one new person in every 75 milliseconds.

The **Check-Out Person** will take money from the customers. They can only record one can of soda every 10 milliseconds. Each can of Pepsi is sold for $.50 and each can of Coke is $.55. Remember that the customers have unlimited money, so you don't need to verify, you just need to record.

At the end of the day the store needs **to reconcile the bill** from each of the delivery people and the **income from selling** soda.

Output: Display to the user in a **nice, formatted** way what is going on in the store. Make sure to include the stock of the shelves and the running totals of the bills/revenue.

Question (10pts): What are your bottlenecks in your code? How can you identify them? Are they fixed or variable?

Question (10pts): How would you change your code to move the bottleneck to the delivery persons? What would you change to move the bottleneck to the check-out person? Test your code, make sure it can handle different modes of operation.

Reference Code:

Go routine sleeping:

```
import "time"

func main(){

	time.Sleep(100 * time.Millisecond)
	//This causes the current routine to sleep for 100 Milliseconds

}
```

Enumerations:

```
type Brand int //Sets the type Brand to represent integers

const (
	Coke Brand = 0		//Sets 'Coke' Brand to be equal to 0
	Pepsi Brand = 1		//Sets 'Pepsi' Brand to be equal to 1
)


func main(){

	var kind Brand

	kind = Coke

	if (kind == Pepsi){

		fmt.PrintLn("Not Coke!"

	}

}
```

Random Variables:

https://golang.org/pkg/math/rand/#Rand