**Semaphore and multi-threaded Simulation Program Design Document**

**By Timothy Snipes**

**Introduction**

This design document describes a semaphore Java program that will use the semaphore class features to simulate, and synchronize a burrito ordering program. The program will simulate multiple thread processing, and use the semaphore class to control the concurrency of the threaded program processing methods. The Burrito Brothers program must only use the internal semaphore class methods to synchronize customer ordering/waiting times.

**Methodology**

The “Burrito Brothers” JAVA program will perform the following functions:

1. Simulate parallel processing threads using Java semaphore class.
2. The semaphore methods will control the concurrency of the Burrito Brothers order processing.
3. The solution will create methods and classes that must be executed from the command line/terminal interface.

*Goals and Strategies*

The stated purpose of this project is to simulate a threaded processing chain for a restaurant. The intent is to highlight parallel processing that occurs within a system’s CPU. Semaphores are used to constraint or prevent the number of threads that can accessresource some physical or logical resource.

Semaphores work by maintaining a set of permits, that’re necessary to block other permits from executing. A way in which this program will simulate semaphore blocks, is by using the Semaphore class to keep count of the number of available tasks respectively. This project will create methods to count, restrict, and redistribute tasks for execution.

*Constraints*

This project relies heavily on the Java SE Semaphore class. The following mandatory constraints are applied to the running program:

1. *Store* can have a maximum of 15 customers from *Customer* class*.*
2. Each customer has an order of 1 to 25 items.
3. Arrival time is calculated in milliseconds.

*Technical Considerations*

This project will run on Java SE 7. It must be ran from the command line using javac. The minimum system requirements to run a Java program: CPU 2 GHZ, RAM 512 MB, minimum 10 GB free space HD, video card 128 MB.

*Reliability*

The program runs every time it’s ran. The system will cause some instability if the max settings are surpassed.

\*The program still continues to run if settings are surpassed, although less stable.

*Validity*

Thread priority is divided into tasks. The tasks’ priorities are validated by the semaphore class and methods.

*Description of Repeatable Steps*

The program’s intent is to simulate a restaurant’s check-in/out procedures. The following are the steps that’re constantly executed repeatedly:

-Customer arrivals

-Server is serving customer or waiting.

-One patron must pay at a time.

-Customers enter waiting area until order is complete.

-Waiting area is organized by shortest order completion time.

-Register processes customer orders.

-After the order is processed, the customer leaves the restaurant.

-If no more order are in the waiting queue, the program exits.

**Design**

SCOPE/OVERVIEW OF “BURRITO BROTHERS”

The program will be broken down into its respective classes and their methods. The solution must address threading, through the Java’s semaphore class. The program will implement the semaphore class to dictates task priorities of the ‘Burrito Brothers’ restaurant. User input can be designated automatically or manually.

CUSTOMER STEPS FOR EXECUTION

A user will enter the store, input burrito order details, a server will take the order to a register, process the order, and deliver the order to customer. The program will end after all customers are no longer waiting for orders.

The following are the class definitions:

**Burrito** – Prompt for user input(User Mode) or set amount of burritos requested.

**Customer** – Contains customer ID information and burrito order size.

**Server** – Contains server information for checking customers in and out of the store.

**Store** – Contains store information, store capacity, and implements semaphore class for threading of tasks.

Architectural Design

UML:

**Burrito**

#int NoOfCustomers

#int NoOfServers

#int StorCapacity

#int ArrivalWindowMax

#int CookingSpeed

#int RegisterSpeed

#Boolean advmode

+main()

-int input()

-int input

-string skip

-boolean flag

**Server**

-int serverID

-Customer atCounter

-int ServersInStor

+Server(int serverID)

+FreeServer()

+run()

**Store**

**-**Store Str

-private int CustInShop

#int CustInLine

#int custID

#Customer line

#LinkedList<Customer> RegisterLine

#Semaphore semCustInStore

#Semaphore semStartServing

#Semaphore semRegister

#Semaphore semRegisterLine

#Semaphore semCounter

#Semaphore semLine

#Semaphore semIngredients

+Store getShop()

+Welcome()

+CheckLine(Customer Cust, Boolean newCust)

+EnterLine(int CustInLine, int Sortuntill, Customer Cust)

+Customer Counter(int serverID)

+Customer atCounter

+String Line

+Cooking(int burritos, int ServerID)

+GoToPay(Customer atCounter)

+ LinkedList<Customer> RegisterPrint

+String AtRegister

+Register()

+Customer cust

+run()

**Customer**

-int custID

-int orderSize

+Customer()

#int getOrderSize()

#redOrderSize()

+int getCustID

+ setCustID(int custID)

Top-level ERD:



Flow Chart:



Works Cited

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts*(9th ed.). Hoboken, NJ: Wiley.