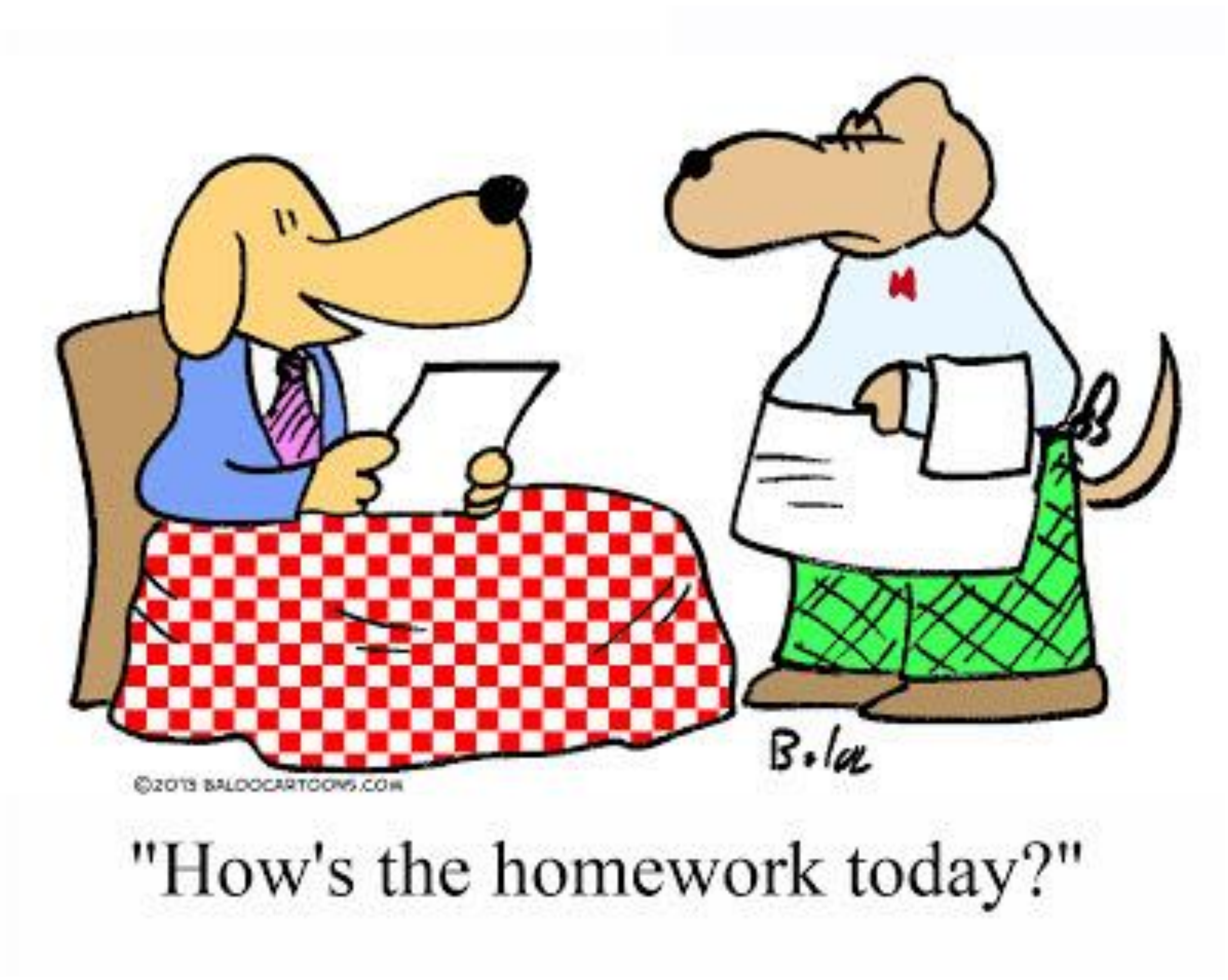
SWE: Serve with Ease

GROUP #1
Project: Restauranting Made Easy
Technical Documentation



Team Members: Annie Antony, Athira Haridas, Christina Parry, Emma Roussos, Christina Segerholm, and Nishtha Sharma

332: 452 Software Engineering, Professor Ivan Marsic

Github URL: https://github.com/powerpuffprogrammers

Table of Contents:

1. Overview of the System
2. Packages
3. Javadoc
4. Communicating between Processes

1.) Overview of the System

The SWE system implements a client server structure where each employee with a tablet is a client and the three databases and message controller act as the server. SWE is also implements a distributed database system because it uses three different databases.
The databases are as follows:

- Database A holds all employee information. This will handle login and logout requests from all employees.
- Database B holds the list of tables in the restaurant and the information associated with that table. This will handle requests from the Host to get table data.
- Database C holds the menu, the inventory, and a list of receipts that are already paid. This will handle requests for the menu from the waiter as well as updates on the inventory from the chef. It will also allow for auditing by recording every receipt that gets paid. The waiter will send its paid tickets to the Database.

The Message Controller will be used to forward messages from one employee to another. This way the user's tablet does not have to know all of the other active tablet's IP addresses, only the Message Controller's IP address.

Database A, B, and C, as well as the Message Controller, will run on their own processes. They can be started by running DatabaseAcontroller.java, DatabaseBcontroller.java, DatabaseCcontroller.java, and MessageController.java. All of these are multi-threaded.

Each employee should have their own tablet and this tablet will run TabletApp.java.
This program will set up a connection with Database A and send a request with an employee id to login. The response will include the position of the employee id on success. Then it will close this connection and start up the interface that corresponds to that position.
Each interface has a

- Message listener: constantly listens for messages sent from the Message Controller.
- Message sender: sends any messages from this user to the Message Controller
- Interface Controller: controls the data associated with this interface and what screen is open
- JPanels (one for each screen): draws the screen for the user using the data in the interface controller

There is a host, chef, manager, and waiter interface.

The configuration files are used to hold the IP Addresses and port numbers of the Databases and the Message Controller and must be updated accordingly in configuration/domainNames.txt and configuration/portNumbers.txt.

2.) Packages:

Chef
- Holds all files associated with the chef interface. Notice that this is not finished (labeled WIP = work in process).

Configuration
- Holds files used for configuring the sockets. Holds the text files that hold the IP Addresses and Port Numbers of the DB A, B, C and Message Controller servers.

databaseA
- Holds controller (used to start up DB A) and the data structure class Employee.

dataBaseC
- Holds controller(used to start up DB B) and the following data structure classes: Table, TableList

dataBaseC

- Holds controller (used to start up DB C) and the following data structure classes: Dish, DishData, Ingredient, Menu.

Host

- Holds all files associated with the host interface. (listed in overview of system)

loggingIn

- Holds TabletApp.java and the log in screen panel.

Manager

- Holds all files associated with the manager interface. (listed in overview of system)

messageController

- Holds all files associated with the message controller and the data structure class Message.

Waiter

- Holds all files associated with the waiter interface. (listed in overview of system)

3.) Javadoc

These Docs show describe the function of each method and class within the application.
All Javadocs are attached in folder Javadocs. Each java class has its own doc and the docs are separated by package (noted above).

4) Communicating between Processes:

All of our process communication was done with Java Sockets. In order to get data from one process to another, first we had to encode that data as a string, figure out who to send it to, and then once it go to the other process be able to decode the string to understand the data. To make this easier we made a Message java object that held both the sender's and receiver's id and position. It also held a String for the content of the message. Whenever communication is needed, a message is created by the sender who fills out a Message object and then sends it to the Message Controller to be forwarded. When the message is sent to the message controller the message must be encoded in order to send it across the socket. Since we cannot send java objects across sockets easier we create a toString() to encode the message, and a fromString() to decode the message. This made communication quite simple because once you decoded the message you were left with a message object that held the sender and receiver's info as well as the content.
The content of the message will be different depending on the type of message that needs to be sent. The content of the message could actually be English (this will be used for displaying notifications on the receiver's table screen).
For certain messages, the content would be encoded data.
For the reading the menu from Database C and reading the list of tables from Database B, we used Google's GSON to help us encode and decode the data. For sending Tickets from the waiter to the Chef or from waiter to Database C for auditing, we made our own toString() and fromString() methods for the ticket class.