**IBSL Computer Science**

**Internal Assessment**

**Program Dossier**

**Restaurant’s Reservation System**

Hong Joon Choi

2013.02.01

**Table of Contents**

**A – Analysis**

**A1. Analyzing the problem\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4**

**A2. Criteria for success\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6**

**A3. Prototype solution\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7**

**B-Design**

**B1. Data structures\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_12**

**B2. Algorithms\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_13**

**B3. Modular organization\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_19**

**C-The Program**

**C1. Program code\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_21**

**C2. Handling errors\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_38**

**D-Documentation**

**D1. Annotated hard copy\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_41**

**D2. Evaluating solutions\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_46**

**Stage A : Analysis**

**A1. Analyzing the problem**

**A1.1 Jumbo Seafood restaurant:**

**JUMBO seafood** is a restaurant at Clarke in Singapore that specializes in seafood. The restaurant especially promotes dishes like chili crab. This has many other franchises at different places such as at East Coast, Riverside Valley and Dempsey hill.

**A1.2 Current System:**

Restaurant manages reservation system which keeps track of time of reservation, number of people and any extra requests of customers who booked the restaurant.

When the reservation is made by customer, user adds a reservation on the list. Then, when customers come to restaurant and tell a name to a user, user finds the name in the list and guides them to table that user arranged for them. Then, the list is highlighted to indicate that it has been processed.  
Reservation document includes data of customer’s name, date and time of reservation, number of people and additional requests. Only reservation for 11A.M to 2P.M and 7P.M to 10P.M is accepted. Workers assign table for them, and each table is assigned with a number. Unless requested, customers are given with two hours to finish their meal.

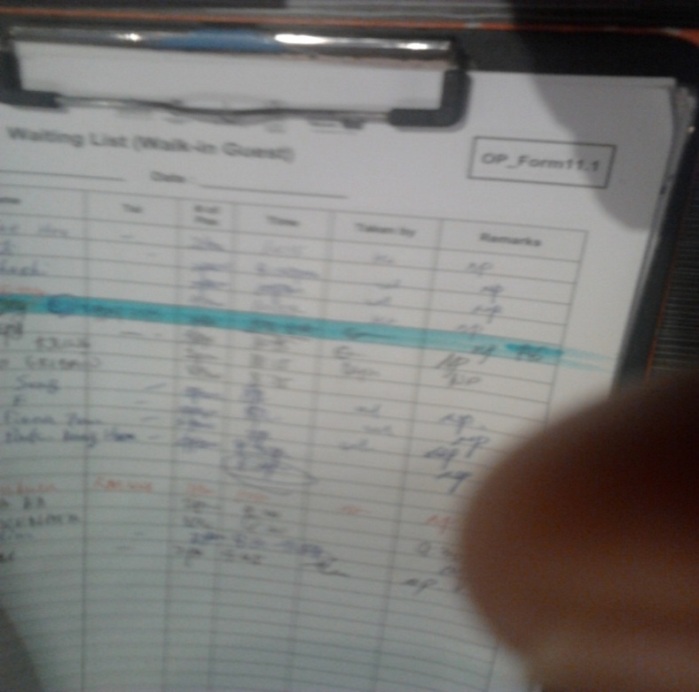


Figure 1

Figure 1 is a picture of current document from the system. User of the system use both computer and paper. However, according to manager of the restaurant, computer only keeps track of table number that is being occupied currently.

**A1.4 End-User:**

User of the system is clerk of the counter in the restaurant, who manages dish reservations.

When I went to the restaurant to see their working places, it seemed that they are very busy at receiving the phone, writing reservation list and striking through for the finished one and mark with colored pen. When I reserved in advance and went to the restaurant to have dinner there, clerks searched for our name and it took about a minute. I interviewed manager of the restaurant and I could get information about current system and problem associated with it.



Figure 2

Figure 2 illustrates how users at the restaurant manage reservation. They are searching for names in the list. They highlight list of reservation which they finished processing.

**A1.5 Problems:**

During the interview with the manager, he mentioned some problems associated with the current system.

1. It is time consuming to search for customer’s name in reservation list because the document has lots of customers and it takes too much time to look through all names to find a customer’s name.
2. Paper document doesn’t allow efficient deleting and sorting of the list of customers by order of time of reservation. User has to highlight the list to indicate that it is in no longer use. So, because the reservation list could get unorganized easily, it will make the user’s job even more time consuming.
3. Readability is another problem. List of reservation is recorded by handwriting and sometimes handwriting is hard to read. It takes more time to correctly read what user had written down.

**A2. Criteria for success**

The goal of this dossier is to build a program that keeps track of restaurant reservation. The new system would focus on solving problems of the current system, which is mainly about time consumption and readability. Program should be able to search, sort and delete reservation list. Also, the program should be able to output reservation list into text file.

**A2.1 Objectives**

|  |  |  |
| --- | --- | --- |
| Objective | Description | Evidence |
| A | Have faster search through customer’s names and their details. | Enter a name to search. Enter \* to select all names.  Input name : joon  two relevant names found.  name | Time | person | table no. | request |  ----------------------------------------------------------------------  Hongjoon | 20/10/12 6:00PM | 4 | 36 | yes |  Joonrock | 22/10/12 11:00AM | 6 | 14 | no | |
| B | Have sorting and deleting of reservation list. List should be sorted according to dates and time. | Hongjoon | 20/10/12 6:00PM | 4 | 36 | yes |  Enter [d] to delete and [s] to skip  Input : d  Successfully deleted and sorted. |
| C | Allow user to add reservation to list, and Reservation list should be easily read. | Enter staff name, wage and year of birth respectively.  Input name : a  Input date : 27/9/12  Input time : 12  Input number of people : 4  Any request? : y  Input request: I want to dine at the table outdoor.  Successfully added to the list and sorted |

**A2.2 User friendly Objectives**

The program should have clear Interface for it to be user-friendly. Menus should be lined up in the middle, important information should stand out so that it is easy to read and reservation document has to be in table format and has to be organized. I will display a table as a reservation list so that it is easier for user to read.

First, program should have organized menu screen and should use clear language.

Secondly, program should require user minimum inputs as possible

Lastly, program should perform effective error handling.

In general, the purpose of the program is to provide user with an effective, efficient and simple way to manage restaurant reservations. This should solve problems with current system discussed in section A1.

**A3. Prototype solution**

**A3.1 Structure diagram**

Main Menu

Save and quit

Display Reservation List

Add reservation

Search for names

View additional request

Edit information

Delete reservation

**A3.2 Prototype**

Main menu will look like the diagram below.

==========Main Menu=========

[d] Display reservation list

[a] Add reservation to list

[s] Search for names

[q] save and quit

User input :

User should enter a character that is in front of each menu. For example, entering ‘d’ will show all lists of reservations. Even if user makes mistake and enter ‘da’ for example, the program will do the same thing.

First menu, Display reservation list will look like this.

Enter date. (DD/MM/YY) Enter any other key to see today’s reservation.

User Input

name | Time | person | table no. | request |

----------------------------------------------------------------------

Hongjoon | 27/9/12 11:00AM | 4 | 36 | yes |

Joonrock | 27/9/12 12:00PM | 6 | 14 | no | Jaehoon | 27/9/12 7:00PM | 3 | 14 | no |

User will be able to see list of reservation at specified date.

Below is a screen of adding reservation,

Enter staff name, wage and year of birth respectively.

Input name : (input String)

Input date : (Input String as DD/MM/YY form)

Input time : (Input String as hour:minute)

Input number of people : (Input integer, less than 3 digit)

Any request? : (Input character, ‘y’ or ‘n’)

Input request: (Input String)

Successfully added to the list and sorted

For inputting hour, only integers between 11~13 and 19~21 will be accepted. If the number is out of range, or if there is an exception in input, exception will be caught and user will be asked to enter it again. Every time when user adds a reservation, the program will automatically sort the list. The program will assign available table for the customer.

Below is an expected display of Searching menu.

|  |
| --- |
| Enter a name to search. Enter \* to select all names.  User input : joon  Two relevant names are found.  name | Time | person | table no. | request |  Hongjoon | 20/10/12 6:00PM | 4 | 36 | yes |  Joonrock | 22/10/12 11:00AM | 6 | 14 | no |  Hongjoon | 20/10/12 6:00PM | 4 | 36 | yes |  Enter [r] to show request, [e] to edit, [d] to delete or [s] to skip  Input : r  I want to dine at the table outdoor.  Joonrock | 22/10/12 11:00AM | 6 | 14 | no |  Enter [e] to edit, [d] to delete or [s] to skip  Input : d  Successfully processed and sorted. |

Display shown above is just an example. user input any word, all names that include that word will appear. Then, user will be asked what to do. If customer had any additional request, user can see it from search menu. After displaying request, and deleting reservation, list will be sorted.

**A3.3 User feedback**

I showed the prototype solution to the user. The restaurant had been using paper based system for a long time, and the user commented that it will be costly to set up a new system. However, he said that computer based system will solve the problem of current system and that it will give much more flexibility and efficiency of work in the long term because it will save lots of time.

The user commented that there are some more things that the program needs to have for the prototype solution. He firstly said that he should be able to manually assign table for customers because they sometimes request for specific places, such as eating outside or inside.

He also said that there are some cases that customers, who reserved for a restaurant, don’t show up. He worried that there could be lists that remain undeleted. Thus, I decided to create function of deleting outdated reservation into main menu screen. Lastly, he said that it is too annoying to quit the program he wants to save the data. He asked me to make a program that saves data automatically.

**A3.4 Improvements in prototype**

1. Add a menu that deletes all outdated reservation lists.
2. Enable user to assign table for customers manually.
3. Save the data whenever the modification to reservation list is made.

**Stage B : Design**

B1. Data structures

The file that holds reservation will be a sequential file, with all the sorted reservations saved. Each reservation will have five fields: customer name, reserved time, number of customers, table number and additional request.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Variable type** | **Sample data** | **justification** |
| CustomerName | String | HongJoon | Customer name is a string because name is made of letters |
| ReservedTime | Calendar | 1352300886780 | Reserved time is represented in milliseconds, Calendar data type has built in function that does conversion for us. Millisecond form will be changed into specific dates and times whenever user has to see this data. |
| numberOfCustomers | byte | 8 | Number of customers will be in integer form, and it won’t require large variable size since it is not possible for groups of 100 people to reserve a restaurant. |
| TableNumber | Int[] | 74, 75 | Each tables is assigned with its ID number, and it has four seats on it.  Variable type is in array because more than two tables may be assigned for group of customers who have more than 5 people. |
| request | String | I want to seat outdoor | Because request from customer is in sentence form, String variable is used. |

An array of reservations at Restaurant class will read all the bookings for specific day and it will list on screen. When the user wants to get specific reservation for restaurant, program will search for names. When a name is found, program will allow user to edit or delete the booking when it is processed.

B2. Algorithms

|  |  |  |
| --- | --- | --- |
| **Name** | Search | |
| **Description** | Returns index of array that holds the requested String | |
| **Preconditions** | There is an array that holds data. | |
| **Parameters** | **Local Variable**  **Name Type Vaulue** | **Return values** |
| String ss  String[] array |  | Int |
| **Code** | for int i=0, i<array.length  if ss = array[i]  return i  end if  next i  return -1 | |
| **PostConditions** | Index of an array will be found and will allow user to do operations with it, if nothing is found, -1 is returned | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | | smartSearch | |
| **Description** | | Returns multiple indexes of array that has part of the String. For example, “james” and “amy” will be found if user inputs “a” | |
| **Preconditions** | | There is an array that holds data of String | |
| **Parameters** | **Local Variable**  **Name Type Vaulue** | | **Return values** |
| String ss  String[] array  Int arraySize | list int[] -1  numberFound int 0  disassemble char[] ‘’  reassembled String “” | | Int[] |
| **Code** | for int i=0, i<arraySize  disAssemble = staff[i] to array of characters  for k=0, k<staff[i].length  String assembled =""  for int j=k, j<staff[i].length  assembled+=disAssemble[j]  next j  if assembled startsWith ss  list[numberFound]=i  numberFound++  break  end if  next k  next i  return list | | |
| **PostConditions** | Array of integers which indicate index of the array will be returned. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | | Sort | |
| **Description** | | Sorts array in ascending numerical order. | |
| **Preconditions** | | Array that holds time values (in milliseconds), is out of order | |
| **Parameters** | **Local Variable**  **Name Type Vaulue** | | **Return values** |
| int[] time | Swap int 0 | | int[] |
| **Code** | for k=0,k<length  for int i=0,i<length  if time[k]<time[i]  swap = time[k]  time[k] = time[i]  time[i] = swap  end if  next i  next k  return data | | |
| **PostConditions** | Array will be sorted in ascending order. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | Delete | | |
| **Description** | Deletes an index from array, and decrease size of the array by 1 | | |
| **Preconditions** |  | | |
| **Parameters** | | **Local Variable**  **Name Type Value** | **Return values** |
| int index  Reservation[] original  int arraySize | | overwrited Reservation[] null | Reservation[] |
| **Code** | | for int i=0,i<index  overwrited[i] = original[i]  next i  for int j=index, j<arraySize-1  overwrited[j] = original[j+1]  next j  return staffOverwrite | |
| **PostConditions** | | Data in array at particular index will be deleted and size of the array will be decreased by 1. | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | allignToMiddle | | |
| **Description** | Aligns String to the middle of given amount of spaces | | |
| **Preconditions** |  | | |
| **Parameters** | | **Local Variable**  **Name Type Value** | **Return values** |
| String message  Int space | | ss String “” | String |
| **Code** | | int blank=space-a.length;  if blank%2!=0  ss+=" "  end if  for int i=0,i<(blank/2)  ss+=" "  next i  ss+= message  for int i=0;i<(blank/2)  ss+=" "  next i  return ss | |
| **PostConditions** | | Message is aligned at the center of given space. | |

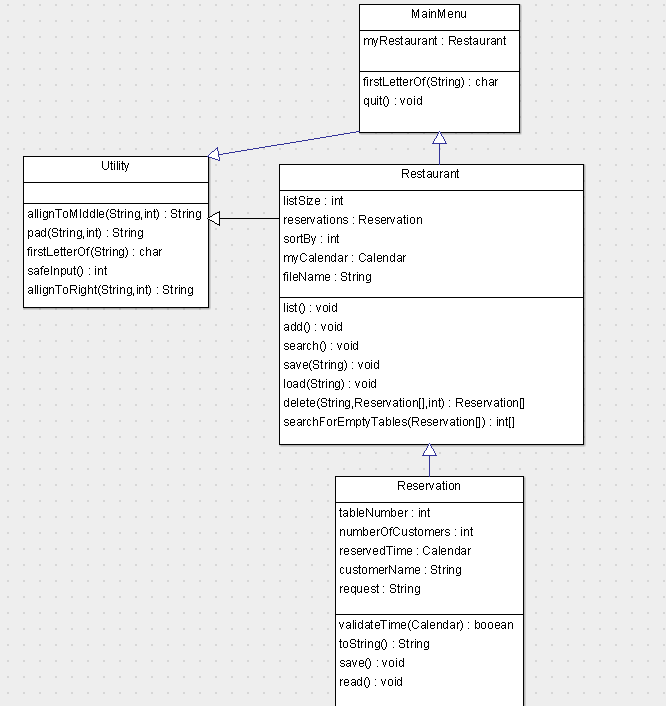
|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | Pad | | |
| **Description** | Allocates String and leaves space until the given points | | |
| **Preconditions** |  | | |
| **Parameters** | | **Local Variable**  **Name Type Value** | **Return values** |
| String message  Int space | | ss String “” | String |
| **Code** | | do  ss+=" ";  while ss.length<space  return ss | |
| **PostConditions** | |  | |

|  |  |  |
| --- | --- | --- |
| **Name** | validateIfTableIsEmpty | |
| **Description** | Validates if chosen table or tables are empty at specific time. The method returns true if the table or tables are available to be used at specific time. Customer who reserved a table is given 2 hous to finish their meal | |
| **Preconditions** |  | |
| **Parameters** | **Local Variable**  **Name Type Vaulue** | **Return values** |
| Long time  Reservation[] app  Int[] tables |  | Boolean |
| **Code** | for i=0, i<app.length  if app[i].reservationTime-time<3600000  for j=0, j<tables.length  for k=0, k<app[i].tableNumber.length  if tables[j]=app[i].tableNumber[k]  return false  end if  next k  next j  end if  next i  return true | |
| **PostConditions** | Program will indicate user if chosen tables are available at specific time | |

B3. Modular organization

**B3.1 class diagrams and explanations**

The program will have four classes, MainMenu, Restaurant, Utility and Reservation.



**Class 1 :MainMenu**

The MainMenu will be responsible for creating the restaurant class. It will also have to save the file, having an array to keep reservations saved. Main menu does not have any function except for quit().

**Class 2 : Utility**

This class is a collection of utility functions. Functions include aligning methods, input method that handles error and method that returns first character of the String. Class MainMenu and Resstaurant extends this class.

**Class 3 : Restaurant**

Restaurant class is collection class of reservations. This class is responsible for reading all reservations made and saved to the file. This class has two fields: reservations and size of reservation list. It has five functions that manages reservations: list, add, search, delete and function that validates if chosen table is empty at specific time. Other functions are save() and load().

**Class 4 : Reservation**

Reservation class is base class of the program and it holds data of one reservation. It has five fields to it, which is described at part B1. It has function of validating the range of time.

**B3.2 linking to solutions to problems.**

**Legibility :** List() function will print out list of reservations in clear and organized way. Utility functions at class Utility, such as AllignToMiddle() and pad() will make the list much clearer.

**Sort :** sorting algorithm will solve the problem of putting list out of order whenever modification is done. The program will automatically sort array whenever any action is done by user.

**Search :** Even if customer doesn’t spell out his/her name clearly, or if spelling of customer’s name is unclear, smartSort() method will allow user to find for names just by entering part of their names.

**Stage C : The Program**