Baran Korkmaz 302809

Restaurant POS System

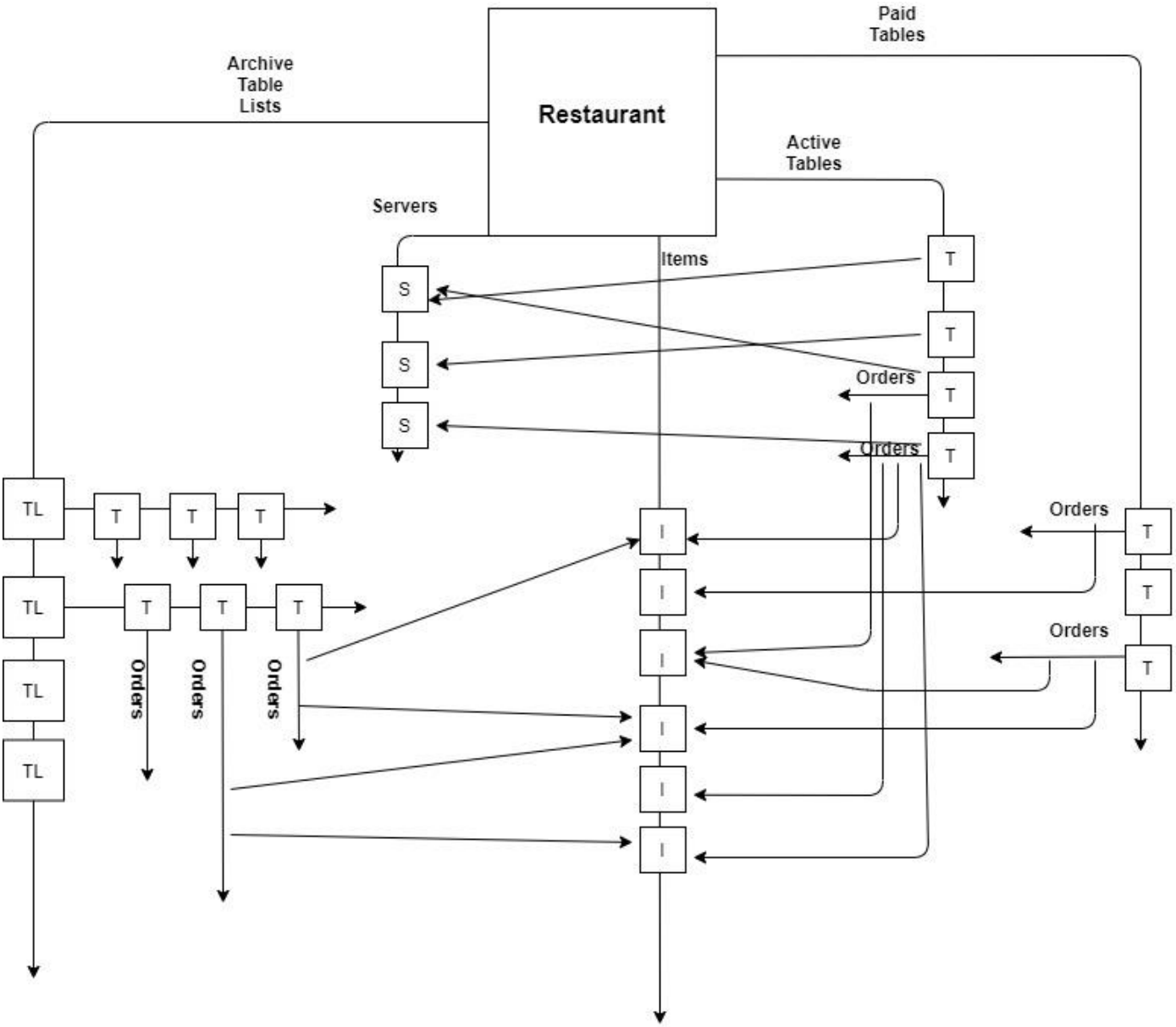
The aim of this program is to provide a user with certain functionalities to manage a restaurant

- Storing the information about menu items. Menu can be updated at any given time.
- Assigning servers to the tables.
- Placing ordered items to the tables.
- Calculating and displaying the total earnings from requested table or all tables from a previous date.
- Keeping track of payment status for each table in.
- Archiving old lists and showing one from a specific date if wanted.

Restrictions of the program are:

- Items added to the tables must be from the ones in the menu.
- All parameters of the restaurant, menu items, servers and tables must be valid (e.g. no negative price, no invalid date, no missing parameters).
- Program won't reset and archive the whole list automatically at any time. If wanted, it must be done manually at the end of the day and only at the end of the day. Dates also must be entered manually.
- Every table can be assigned with only one server. Every table must be assigned with a server before other functions such as putting or removing orders can be used.
- If a table added to the paid list means its bill has been paid and customers has no longer using that table. There is no other that system that checks if a table's bill is paid or not. Therefore, user must be responsible when adding a table to paid list.

Memory Map


Restaurant
Archive Table Lists
Paid Tables
Active Tables
Servers
Items
Orders
S
T
TL
I

Description of Classes

• Class Restaurant

This is the main class of the Project connected to all other classes. The user interacts with this class and all the functionality of the program is accessible through this class.

It contains the general information of the restaurant and its fields. Also contains a vector of pointers to staff, tables, and archived table lists.

```
class Restaurant{
char* restaurant_name;
int amount_table;
Table_List *paid;//table list for tables used throughout the day, paid and left
Table_List *active;//active table list
vector<Table_List*> archive;//vector of pointers to table lists archived
vector<Server*> server_list;//vector of pointers to the servers
vector<Item*> item_list;//vector of pointers to items available
Server *get_Server(int server_no);//gets a pointer to a server
bool exist_server(int server_id);//checks if server exists
bool exist_item(int item_id);//check if item exists
bool check_assigned(int table_no);//checks if a server assigned
Item *get_item_at(int item_id);//returns the item with given id

public:
Restaurant(char *restaurant_name, int amount_table);//creates a restaurant and a list of active table list and paid table list
~Restaurant();
void modify_restaurant(char *restaurant_name, int amount_table);//updating restaurant
void add_server(int server_id, char *server_name);//adds a new server to the server list
void delete_server(int server_id);//removes a server from the server list
void assign_server(int server_id,int table_no);//assign a server to a table
void add_item(int item_id, char* item_name, float item_price);//adds new item to the item list
void delete_item(int item_id);//deletes an existing item from the list
void modify_item(int item_id,char* item_name, float item_price);//modifies an existing item by searching for its id and changing other data
void add_item_to(int table_no, int item_id);//adds an item to a table
void remove_item(int table_no, int item_id);//removes an item from a table
void copy_to_paid(int table_no);//copies an active table and adds it to the paid list, used in clean_table function
void clean_table(int table_no);//copies table to the paid list, cleans the list of orders and assigned server for the table
void end_day(int day,int month,int year);//copies non-empty active table list to the paid list, copies paid list to the archive and cleans both active
void archive_list(int day,int month,int year);//creates a new list from paid list and adds it to archive vector
void delete_archived(int day,int month,int year);//delete a archived list
void total_day();//calculates the total amount of earnings for the active and paid list
void total_archived_list(int day,int month,int year);//total earnings of a day in archive
void total_month(int month,int year);//calculates the total amount of earnings for all of the lists in given month
void total_year(int year);//calculates the total amount of earnings for the given year
void print_server_list();//prints info of the server
void print_active_assignments();//prints the list of servers assignment info
void print_tables();//prints the active tables
```

```
void print_paid_table(int table_no);//prints a table from paid list
void print_paid();//print paid list
void print_items();//prints all the items
void print_table_info(int table_no);//prints the orders of a single table
void print_archived_list(int day, int month, int year);//prints a list from the archive
```

• Class Table_List

Storing the information about table lists and a vector of pointers to tables. Contains functions that manipulate table lists or tables.

```
class Table_List{
int day;
int month;
int year;
vector<Table*> tables;//vector of pointers to the tables in the list
Table *table_at_no(int table_no);//gets the pointer to a table for the given table number

public:
Table_List();
~Table_List();
Table_List(const Table_List & x);//copy constructor
Table_List & operator = (const Table_List & x);//assignment operator
int &refDay();
int &refMonth();
int &refYear();
vector<Table*> &refTables();
void update(int day, int month, int year);
int count_tables();//counts the no of tables in the list
float total_amount();//calculates the total amount of earnings for the list
void copy_all_tables(const Table_List& x);//copying used when copying to a list there are already tables in the list
bool check_if_empty(Table *table);//checks if a table empty or not
void assign_to(Server *server,int table_no);
void print_assignment();//prints assigned servers
void add_table(int table_no);//add new table to the list
void add_table(Table *table);//add a new table to the list
void add_to_table(int table_no, Item *item );//add a item to the table
void print_info_of(int table_no);//print the info of a table
void remove_from_table(int table_no, Item *item);//remove an item from table
void clean_table(int table_no);//clean the items and assigned server from table
Table *copy_of(int table_no);//returns a copy of a table
void print_all();//print the all list
void clean_all();//clean all the list
bool check_table(int table_no);//check if a table empty
float sum_total();//sums up the total earnings from the list
friend ostream & operator<< (ostream& os,const Table_List &x);
```

• Class Table

Storing the information about a table. Contains the list of items ordered by the table as well as information about total cost of all items ordered by the table. Contains a server pointer for the assigned server to the table.

```
class Table{
int table_no;//can have same no since active list will be created beforehand
float table_total;
vector<Item*> orders;//vector of pointers for the items table ordered
Server * spt;//pointer to the server assigned for the table

public:
Table(int table_no);//0 for the total of the table and the status
~Table();
Table(const Table & x);//copy constructor
Table & operator = (const Table & x);//assignment operator
void update_total();//calculate and update the total payment of the table
int &refTable_no();
float &refTable_total();
Server *&refSpt();
friend ostream & operator<< (ostream& os, const Table &x);
void put_order(Item * ipt);//put a new item to the order list
void clean_table();//cleans the list of orders and assigned server, doesn't delete table
void print_info();//prints the info of table
void assign_server(Server * spt);//put the server pointer in to the table
void remove_order(Item *item);//removes an item from order list
```

• Class Server

This is the class holds the information about staff members as well as functions create and destroy servers.

```
class Server{
char* server_name;
int server_id;

public:
Server(char *server_name, int server_id);
~Server();
char *&refServer_name();
int &refServer_id();
friend ostream & operator<< (ostream& os,const Server &x);
```

• Class Item

This is the class for storing information about menu items as well as functions create and destroy items.

```
class Item {
int item_id;
char* item_name;
char* item_type;
float item_price;

public:
Item(int item_id, char *item_name, float item_price);
~Item();
int &refItem_id();
char *&refItem_name();
float &refitem_price();
friend ostream & operator<< (ostream& os,const Item &x);
```

Testing

1. Test creating servers or items with an id that already exists.
2. Test deleting servers or items that doesn't exist.
3. Test creating new items or servers after deletion.
4. Test putting orders to the tables that doesn't exist or doesn't have an assigned server.
5. Test putting orders to the tables with items that doesn't exist
6. Test assigning two servers to one table and assigning servers to tables that doesn't exist
7. Test removing orders from tables that doesn't exist or removing an order that doesn't exist.
8. Test cleaning tables that doesn't exist or already clean
9. Test calculating total that tables or lists that doesn't exist or lists after deletion.
10. Test archiving list with invalid date.
11. Test printing tables or archived lists that doesn't exist.
12. Test deleting archived lists that doesn't exist.
13. Test calculating total earnings from a month or a year after deletion of a list or from a year or month that doesn't exist.