Object Design Document – Version 1

Pizza Restaurant Management System

Software Industry Code Masters (SICM)
Bryson Payne, Gulsah Altun, Curtis Davis, Nishant Trivedi, & Rutwik Trivedi

Submitted in Partial Fulfillment of

CSC 4350/6350 – Spring 2002

Dr. Erdogan Dogdu, Professor

Georgia State University

Revision History:
Version ODD v1, 04/02/02, SICM team, created.

Preface:

This document specifies the Object Design of the Pizza Restaurant Management System. The intended audience for this document is the project team.

Target Audience:

Developers

SICM Members:

Bryson Payne

Curtis Davis

Nishant Trivedi

Gulsah Altun

Rutwik Trivedi

1. Introduction

Pizza Restaurant Management System (PRMS) is designed to accurately and completely model the necessary computer functionality to manage the entire "front end" of a pizza restaurant – including employee timekeeping, scheduling & payroll, customer food and beverage tickets, kitchen and bar orders, and sales reporting.

The objective of the system is to be simple, but also to cover every detail and function needed by a pizza restaurant. PRMS will include a simple user interface that allows employees to clock-in, send customer orders to the kitchen and/or bar, total up customers' checks for payment, and display total sales for the night, as well as clock-out. The system will feature such useful items as keeping track of order/delivery time and splitting checks among large parties.

The system will provide a Graphical User Interface (GUI) and will run on a single computer (with optional printer). Sales, payroll hours, etc., will be stored in database files for reporting.

The subsystem decomposition from which the systems and objects in this document were developed is summarized in the figure below:

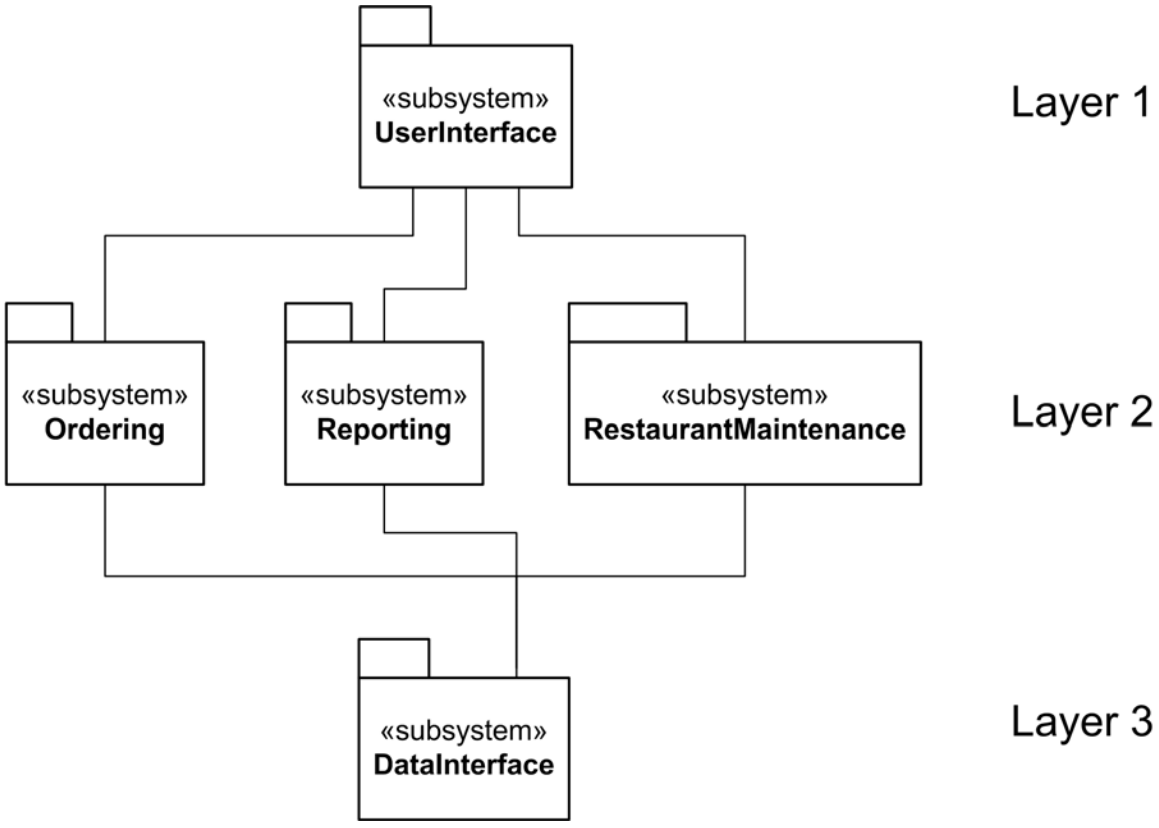


Figure 1-1 Subsystem Decomposition Overview

1.1 Object design trade-offs

As with any software system, a number of explicit trade-off decisions had to be made in the object design. First, the decision to build versus buy components for this system was prescribed to us in the project assignment – all code in this system is our own original work, including data and user interfaces. Second, we struck an initial trade-off agreement on the memory space vs. response time problem by agreeing to put all data on disk (for security in case of power outages, etc., for easy recoverability) rather than keep it in memory. If it is found that response time lags longer than 2 seconds from user entry, this need will be addressed by maintaining a set of memory data structures containing all active orders and employee information as necessary.

For the delivery time vs. functionality question, we have decided to build all subsystems as text-based, menu driven, subroutines. This will allow us to present a prototype in two weeks as required, with most of the functionality of the entire system. The GUI will be added on top of that system so that the final project is realized. This has been a change from our original stance on the rapid prototyping issue listed in our SDDv2, in which we stated that we would withhold the program until fully functional. Due to the requirements of the course, we have changed this trade-off decision.

1.2 Interface documentation guidelines

We have adopted specific interface documentation guidelines and conventions in order to make communication more uniform among the development team. First, all classes are named with singular nouns or noun phrases, with each word beginning with a capital letter. All objects derived from these classes (all class instances) will be named with a similar convention, but with lowercase initial lettering for the first word (similar to the Java convention). Methods are named with verb phrases; fields and parameters are named with noun phrases. Error status is returned via return values in the C++ coding of this system – usually in the form of a Boolean status variable.

Each class prototype will be included at the top of the header file before implementation of that class, and each class' full interface information will be specified in comments with the prototype. Furthermore, each appropriate class will have fully commented pre-condition, post-condition, and invariant information, and every class will have full interfacing information commented before the implementation of that class, including all attributes and methods.

1.3 Definitions, acronyms and abbreviations

RAD **:** Requirements Analysis Document
GUI **:** Graphical User Interface

POS : Point of Sale
PRMS : Pizza Restaurant Management System
SDD : System Design Document
ODD : Object Design Document
SICM : Software Industry Code Masters

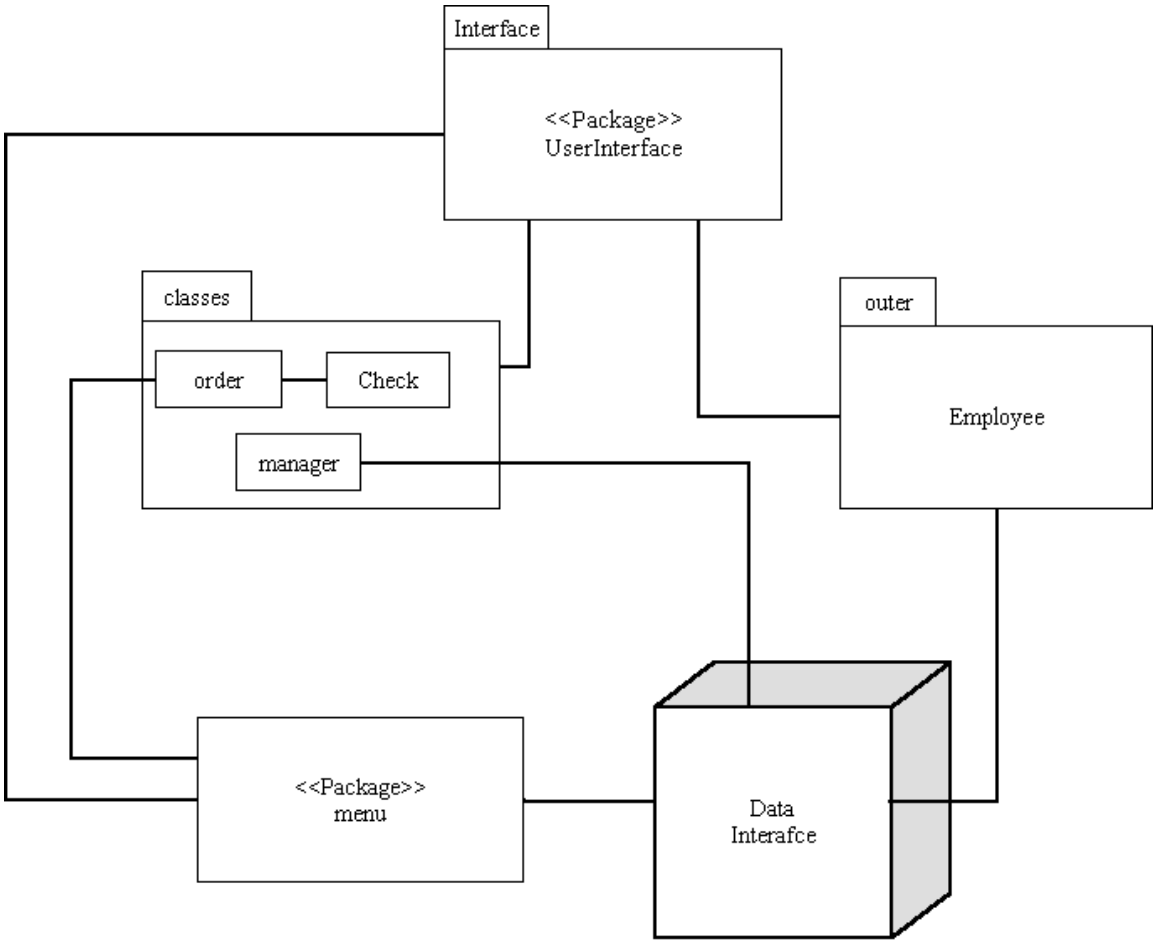
1.4 References

Systems similar to PRMS are being used by most establishments in the industry. Two of our team members have first-hand experience with fast-food restaurant management systems, and two other team members have first-hand knowledge of family restaurant and bar management systems. We have drawn on this experience to produce a system that is more user-friendly and intuitive than the average software available to a small pizza restaurant.

Additional references:

- Version RAD v3 , 02/21/02 SICM team, revised template:
  This document defines the purpose of the system and lists all the functional and nonfunctional requirements given by the client. System models of the system (scenarios, use cases and object models) are listed in detail.
- Version SDD v2, 03/25/02 SICM team, revised template:
  This document defines the subsystem decomposition and system design for the PRMS system. All subsystem design in the ODD is taken directly from our SDD.
- Object-Oriented Software Engineering, Bruegge and Dutoit, Prentice Hall, 2000, page 277. This is the textbook for CSc 4350/6350 from which the template for this document was taken.

2. Packages



2.2. Package Definitions

2.2.1. Interface Package

2.2.1.1 The file organization of code for Interface Package.

This package will include following files:

`UserInterface.h`

This file will contain the definition of each functions needed for the package use.

`UserInterface.cpp`

This file will call those functions for their use.

Note: The functions of each class are described in the class interface section of this report.

2.2.1.2 Overview of Interface Package.

Most of the user interface will be GUI. The user interface accesses all the other packages except the data interface. It is connected tot the data interface through the functions in the classes. Some of the user interface does not lend itself to GUI and will be input as text. For example, the entering of the check numbers and tips by the server will be input as text. All other input will be GUI.

2.2.1.3 Dependencies with other packages.

The interface package is not dependent on the other packages. All the other packages will access interface package to display different kind of menus since it will have all the user interface functions.

The interface package will allow outer user to interact with the system and its functionalities.

The interface package will hold the menus for order, check, and manager classes. Some of the database files such as sales menu, schedule menu and payroll menu will be accessible from interface package.

The menu will access interface package to display menu for food, drink and pizza.

2.2.1.4 Usage of Interface Package.

The interface package will contain the entire GUI so its most important use is to provide menu to all other packages and let other packages interact with each other with those menus. The system will access different areas by menus and interface package will provide those menus.

2.2.2. PRMS classes package

2.2.2.1 The file organization of code for Classes Package.

This package will include following files:

`Order.h`

This file will have definitions of all the functions for the order class.

`Manager.h`

This file will have definitions of all the functions for the manager class.

`Manager.cpp`

This file will call the functions of the Manager.h in order to execute the functionalities of manager.

`Check.h`

This file will have definitions of all the functions for the check class.

`Check.cpp`

This file will execute function of Check.h to perform check functions and process customer requests.

`DailySales.h`

This file will have definitions of all the functions for the DailySales class. This will be used by Manager.h

`WeeklyPayroll.h`

This file will have definitions of all the functions for the WeeklyPayroll class. This will be used by Manger.h

Note: The functions of each class are described in the class interface section of this report.

2.2.2.2 Overview of Classes Package

The PRMS classes package consists of check, ordering and manager. One check can have multiple orders.

The check class will interact with the order class to process the customer's order. It will accomplish this by accessing the order class, which will access the menu package through the ordering class and display beverage and pizza menus. The functions open check, close check and add order are some examples of how the check class will interact with the database to store information about each tables check. To achieve this interaction the check will use the table number to specify which check will be accessed.

The ordering class adds pizzas and beverages to the order. Once the employee presses the "finish" key, the order is appended to the proper check. Also the current order is sent to the proper printer (in our case file) either kitchen or bar.

The manager Class has more interactions with the database tables since the manager will be modifying and updating the files daily sales, weekly sales and weekly schedule. These files are stored in the database. The manager looks at weekly sales. The manager also modifies the weekly schedule. Another functionality of the manager class is to view and print the payroll. This is also accessed from the database

2.2.2.3 Dependencies with other packages.

This package will get the commands from the outer user (employee) through the user interface package. When employee enters to process a customer request he or she will first accesses the check class. Afterwards to take an order the order class will be used to

access the menu and process the order. Once the check is closed it will be stored in the database. The check and order class are accessed sequentially therefore there is line connected to it. Menu is accessed by the order class. Only the manager class has authority to view and modify the items in the database. Other classes are only allowed to store required information in their areas of the database.

2.2.2.4 Usage of Classes Package.

This class used to process customer orders. First the check class is accessed then order and menu packages follow to complete the order. The manager class interact with the database to view/modify the information.

2.2.3. Outer Package

2.2.3.1 The file organization of code for Outer Package.

`Employee.h`

This file will contain all the functions needed for the employee class in order to perform its functionalities.

`Employee.cpp`

This file will call and execute the functions of the Employee.h

Note: The functions of each class are described in the class interface section of this report.

2.2.3.2 Overview of Outer Package.

Employee is connected to the manager through the data interface for several reasons. One, the void function used by the manager corrects and updates a check used by an employee and stored in the data interface. The employee also clocks in and out updating the payroll interface. This database is called by the payroll function used by the manager. Thirdly, the manager modifies the schedule database that can be viewed by the employee.

2.2.3.3 Dependencies with other packages.

This file will interact with classes through interface package in order to perform customer requests. It will interact with Data Interface package to store clock-in/clock-out information and to update payroll, view schedule etc functions related to the employees. Outer package is for any user operating the system, since manager is an employee; he or she will also be in outer class (described as employee).

2.2.3.4 Usage of Outer Package.

This package will be used to perform activities by outside user and providing a way for user to interact with the system. In PRMS this user is employee. Employee will handle most of the functionalities of the system. He or she will be responsible for taking orders, processing checks, and operating other valuable tasks. Since the manager is also an employee he or she is also included in this class. Manager will access the system with proper authentication so, for him or her, the different kinds of options will be provided by the user interface class, distinguishing him or her from the rest of the employees.

2.2.4 Menu Package

2.2.4.1 The file organization of code for Menu Package.

`FoodMenu.h`

This file will have definitions of all the functions for the FoodMenu class. The file will be used by Order.h

`DrinkMenu.h`

This file will have definitions of all the functions of the DrinkMenu class. The file will be called by Order.h

`pizzaMenu.h`

This file will have definitions of all the functions of pizzaMenu class.

`beverageMenu.h`

This fill will have definitions of all the functions of BeverageMenu class.

Note: The functions of each class are described in the class interface section of this report.

2.2.4.2 Overview of Menu Package.

This package will contain the menus for food and drink. It will be accessed by the order class to show pizza and beverage menus for taking the orders. The menu will contain functions for Foodmenu and DrinkMenu for manager to modify those menus.

This package will mostly interact with the classes package since all of the functionalities are related to the classes. It is accessed and modified by the classes that are in the classes package of the system.

2.2.4.3 Dependencies with other packages.

This package will be dependent on the classes package. The classes will be used to access this package. The manager will be able to modify food and drink menus in this package. The check class accesses the order class, which will go to pizza and beverage menus to get the order. These menus are in the menu package. So, order will access menu package each time when processing customer orders.

2.2.4.4 Usage of Menu Package.

The menu package will be used to provide pizza and beverage menus to order class. These menus will be accessed to process the orders. The menu package will help manager modify the menus for food and drink.

2.2.5 Data Interface Package

2.2.5.1 The file organization of code for Data interface package.

`DataInterface.h`

This file will have definitions of all the functions in the DataInterface class.

Note: The functions of each class are described in the class interface section of this report.

2.2.5.2 Overview of Data Interface Package.

This file will hold information about all the files stored in the database. When ever manager needs information on any files this package will be accessed. Only the manager will have authority to access it. This package will have be able to show total numbers of files in database, report numbers in database, finding items in database, and delete and add items to the database.

2.2.5.3 Dependencies with other packages.

This package will be dependent on manager to receive its commands. The menu will be stored in the database files as well as employee information. All of the packages will have interaction with the data interface package because it holds most important files of the system. Only manager with the proper authentication will be able to access and/or modify the any of the files from this package.

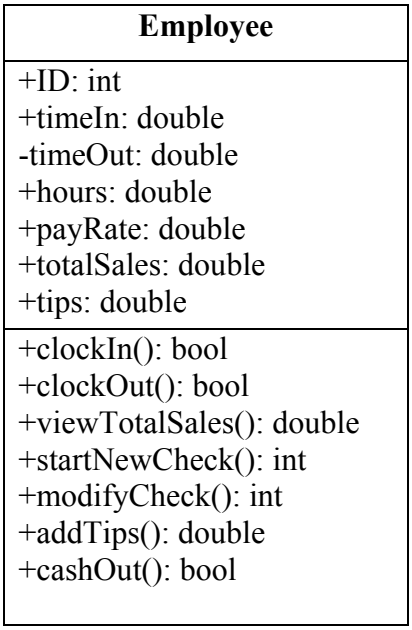
2.2.5.4 Usage of Data Interface package.

Since this package holds the files of the entire system, its most important use is to access and modify the existing files in the database. All of the files needed to store the information about the system will be in the data interface package. It is one of the most important packages of the system.

3. Class Interfaces

3.1 Employee Class

The employee class allows an employee to perform certain operations in order to run the system. An employee can start a new check, modify an existing check, add tips etc. The Employee class is as follows:



Class Name: Employee

Attributes:

a) +ID: int

ID is an employee identification number of type positive integer. It is a public attribute. It is used in order to authenticate the employee with the system.

b) +timeIn: double

timeIn is a value of the time of type double. It is a public attribute. When an employee clocks-in to the system, the time-in attribute is initialized by the value of time at which the employee clocks-in.

c) –timeOut: double

timeout is a value of the time of type double. It is a private attribute. When an employee clocks-out of the system, time-out variable is initialized.

d) +hours: double

hours is a value of type double. It is a public variable. It holds the value of the number of hours an employee works per day.

e) +payRate: double

payRate is a value of type double. It is a public variable. It holds the value of the amount of money an employee earns per hour.

f) +totalSales: double

totalSales is a value of type double. It is a public variable. It holds the value for the total amount of sales for pizza and beverage sales.

g) +tips: double

tips is a value of type double. It is a public variable. It holds the value for the amount of money received as tips.

Operations:

a) +clockIn(): bool

clockIn() is a public function which returns a value of type boolean. When an employee starts job, he or she must clock-in in the system. The clockIn() function records the time an employee started his or her job.

b) +clockOut(): bool

clockOut() is a public function which returns a value of type boolean. When an employee finishes job, he or she must clock-out of the system. The clockOut() function records the time an employee finished his or her job.

c) +viewTotalSales(): double

viewTotalSales() is a public function which returns a value of type double. When this function is called, the amount of total sales, which includes sum of pizza sales and bar sales, is displayed on the screen.

d) +startNewCheck(): int
startNewCheck() is a public function which returns a value of type integer. When this function is called, a new check is started.

e) +modifyCheck(): int

modifyCheck() is a public function which returns a value of type integer. When this function is called, the check that is to be modified, is displayed for modifications.

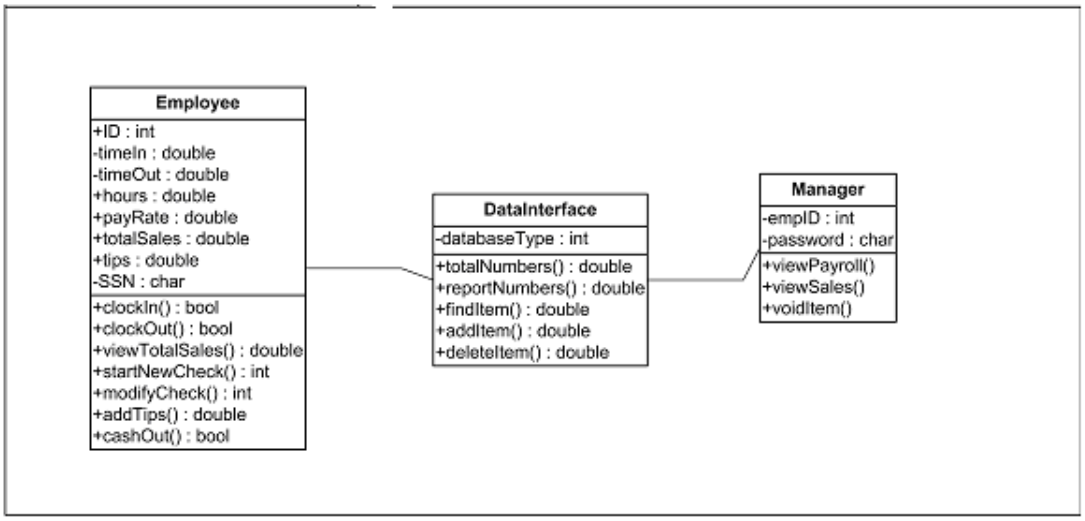
f) +addTips(): double

addTips() is a public function which returns a value of type double. When this function is called, the employee adds the amount of tips he or she receives.

g) +cashOut(): bool

cashOut() is a public function which returns a value of type boolean. When this function is called, the employee can give the amount of change to the customer.

Dependencies with Other Classes and Packages:



As we can see in the figure above, the Employee class interacts with DataInterface class and Manager class, and with ordering package. The employee class lets employees clock-in and clock-out. Once the employee chooses to clock-in, he or she is asked to enter his or her employee number. Once the employee is logged-in to the system with proper and approved identification he or she can access the Ordering package and will be able to assist customers with their orders. When the shift of an employee ends, he or she has to clock-out. At the time of clock-out, the employee enters his or her employee number and the amount of tips earned into the system. After cashing out with the manager, the employee's sales for the day will be totaled to the system, and the employee's information will be saved to the payroll database.

Exceptions Raised and Exception Handling:

There are several exceptions that can be raised by the employee class. For example, if the employee identification number is incorrect or if it does not match with the password, an exception is raised. To handle this exception, an error message is displayed on the screen, and employee is asked to re-login into the system.

3.2 Manager Class

The manager class allows the restaurant manager to view employee payroll, daily sales, or to void an item. The manager class is as follows:

Manager
-empID: int -password: char
+viewPayroll() +viewSales() +voidItem()

Class Name: Manager

Attributes:

a) -empID: int

empID is an employee identification number of type positive integer. It is a private attribute. It is used in order to view employee payroll.

b) –password: char

password is a name of type character string. It is a private attribute. It is used to uniquely identify an employee or the manager with the system.

Operations:

a) +viewPayroll()

viewPayroll() is a public function which does not return any value. When this function is called, the weekly payroll of all the employees is displayed.

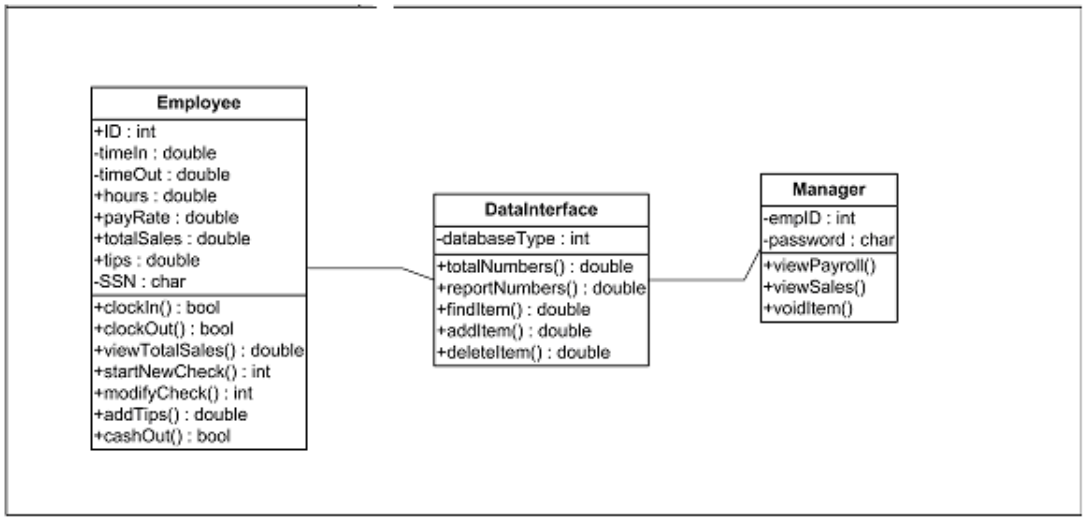
b) +viewSales()

viewSales() is a public function which does not return any value. When this function is called, the daily sales are displayed.

c) +voidItem()

voidItem() is a public function which does not return any value. When this function is called, the manager can void an item from the system.

Dependencies with Other Classes and Packages:



As we can see in the figure above, the manager class interacts with DataInterface class and Employee class, and with UserInterface, Reporting, RestaurantMaintenance, and DataInterface packages. The Manager can view employee payroll, daily sales, or void an item. In order to carry out these operations, the manager class interacts with DataInterface class. The Manager class uses DataInterface class to view employee files, sales files, and payroll files from database. The Manager class does not directly interact with classes such as DailySales, WeeklySchedule, and WeeklyPayroll but it uses DataInterface class to access information regarding those classes.

Exceptions Raised and Exception Handling:

If Manger class attempts to fetch or view data that does not exist in database, an appropriate error message is displayed.

3.3 DailySales Class

The DailySales class allows the restaurant manager to view total sales, pizza sales, and bar sales. The manager can also reset sales. The DailySales class is as follows:

DailySales
+date: char +totalSales: double +barSales: double +pizzaSales: double
+showTotalSales() +showBarSales() +showPizzaSales() +resetSales()

Class Name: DailySales

Attributes:

a) +date: char

date is a value of type character string. It is a public attribute. It is used so that the manager can view any sales record for the given date.

b) +totalSales: double

totalSales is a value of type double. It is a public attribute. It holds the amount of total sales which constitutes pizza sales plus bar sales.

c) +barSales: double

barSales is a value of type double. It is a public attribute. It holds the amount of total bar sales.

d) +pizzaSales: double

pizzaSales is a value of type double. It is a public attribute. It holds the amount of total pizza sales.

Operations:

a) +showTotalSales()

showTotalSales() is a public function which does not return any value. When this function is called, the amount of total sales, which is the sum of pizza sales and beverage sales, is displayed.

b) +showBarSales()

showBarSales() is a public function which does not return any value. When this function is called, the amount of bar sales is displayed.

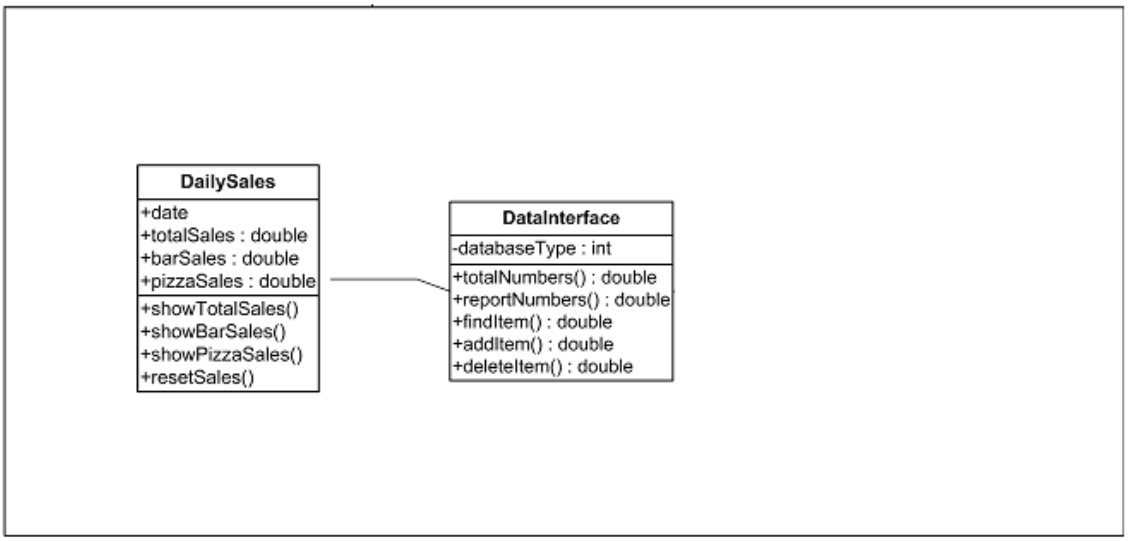
c) +showPizzaSales()

showPizzaSales() is a public function which does not return any value. When this function is called, the amount of pizza sales is displayed.

d) +resetSales()

resetSales() is a public function which does not return any value. When this function is called, the manager can reset pizza sales, bar sales, or total sales.

Dependencies with Other Classes and Packages:



As we can see in the figure above, the DailySales class interacts with DataInterface class and with DataInterface package. The DailySales class displays pizza sales, beverage (i.e. bar) sales, and the total sales for the current day. It can also reset all sales to zero. It interacts with DataInterface class in order to store sales data or to display data that is already stored.

Exceptions Raised and Exception Handling:

If any data item requested by the DailySales class does not exist in the database, then an exception is raised. In order to resolve this, an error message is displayed and the manager is notified which items does or does not exist in database regarding daily sales for the current day.

3.4 WeeklyPayroll Class

The WeeklyPayroll class allows the restaurant manager to view employee payroll for the given week. The WeeklyPayroll class is as follows:

WeeklyPayroll
-totalHours: double -totalTips: double -totalPayroll: double
+printPayroll() +resetPayroll()

Class Name: Weekly Payroll

Attributes:

a) -totalHours: double

totalHours is a value of type double. It is a private attribute. It holds the value of number of hours and minutes that an employee worked.

b) -totalTips: double

totalTips is a value of type double. It is a private attribute. It holds the value of amount of tips an employee receives.

c) -totalPayroll: double

totalPayroll is a value of type double. It is a private attribute. It holds the value of amount of money earned by an employee for a given week.

Operations:

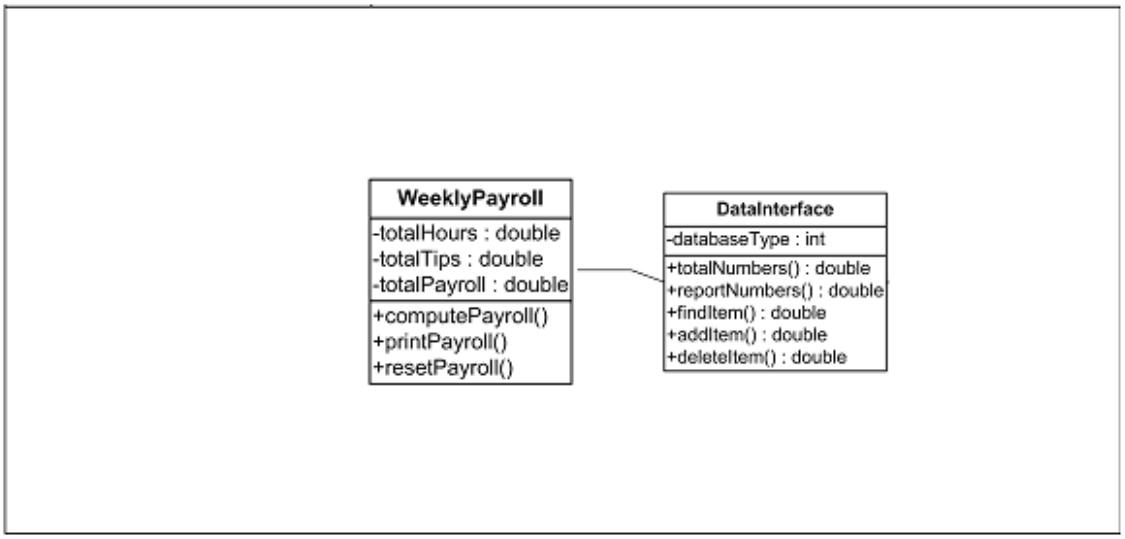
a) +printPayroll()

printPayroll() is a public function which does not return any value. When this function is called, the employee payroll is displayed on the screen. Also, it is stores in database.

b) +resetPayroll()

resetPayroll() is a public function which does not return any value. When this function is called, the employee payroll is reset to all zero amounts.

Dependencies with Other Classes and Packages:



As we can see in the figure above, the WeeklyPayroll class interacts with DataInterface class and with DataInterface package. It computes payroll and stores it in database. Whenever manager wants to see the weekly payroll, he or she can access it through DataInterface class. The printPayroll() operation also fetches data from database and prints it on the screen. The resetPayroll() operation sets all payroll to zero for each employee and stores all the data in payroll files.

Exceptions Raised and Exception Handling:

An exception occurs if manager attempts to access payroll data which has not yet been created. In this situation, an error message is displayed on the screen which notifies manager that employee payroll does not exist in database.

3.5 Class Name: WeeklySchedule

The WeeklySchedule class allows the restaurant manager to view employee schedule for the given week or to reset the employee schedule. The WeeklySchedule class is as follows:

WeeklySchedule
-numberOfEmployees: int -numberOfShifts: int
+setSchedule() +viewSchedule()

Class Name: WeeklySchedule

Attributes:

a) –numberOfEmployees: int

numberOfEmployees is a value of type positive integer. It is a private attribute. It holds the value of total number of employees employed in the restaurant.

b) –numberOfShifts: int

numberOfShifts is a value of type positive integer. It is a private attribute. It holds the value of total number of shifts ran by the restaurant.

Operations:

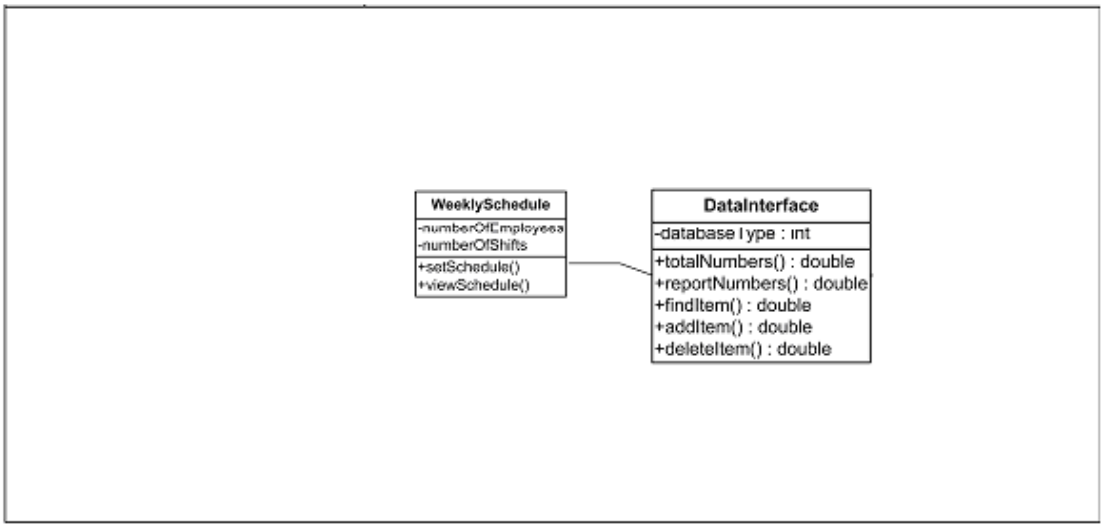
a) +setSchedulel()

setSchedule() is a public function which does not return any value. When this function is called, the manager can create weekly schedule for the employees or modify an existing schedule. This function also stores schedule in the database.

b) +viewSchedule()

viewSchedule() is a public function which does not return any value. When this function is called, the manager can view employee schedule for the current week.

Dependencies with Other Classes and Packages:



As we can see in the figure above, the WeeklyPayroll class interacts with DataInterface class and with DataInterface package. The manager can set employee schedule or view an existing schedule for the current week. The WeeklySchedule class can be accessed or modified only through DataInterface class. No other class can directly access or modify it.

Exceptions Raised and Exception Handling:

If the weekly schedule data file does not exist in database and if manager tries to access or modify it, then an exception occurs. At this point, the manager receives an appropriate error message on the screen.

3.6 Class Name: Coupon

The coupon class allows the restaurant manager to apply coupon on a ticket. The coupon class is as follows:

Coupon
-amount: double
+applyCoupon: double

Class Name: Coupon

Attributes:

a) –amount: double

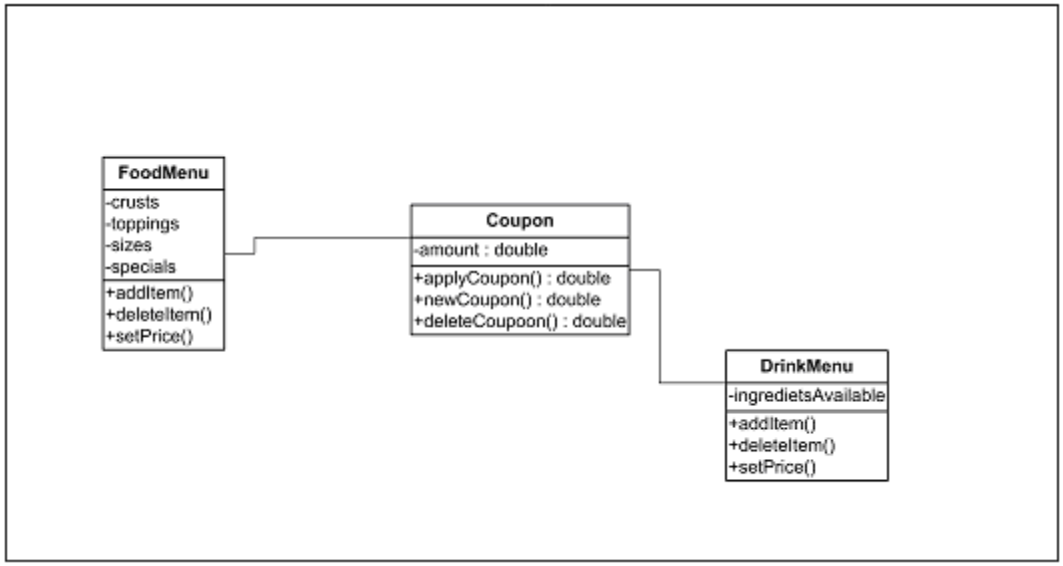
amount is a value of type double. It is a private attribute. It holds the amount of money that is to be subtracted from the check.

Operations:

a) +applyCoupon(): double)

applyCoupon() is a public function which returns a value of type double. When this function is called, the manager can apply a coupon given by the customer on a check. Applying a coupon on a check reduces the total amount of money a customer needs to pay for his or her order.

Dependencies with Other Classes and Packages:



As we can see in the figure above, the Coupon class interacts with DrinkMenu class and FoodMenu class and with RestaurantMaintenance package. If a customer wants to use his or her coupon on a check, then Coupon class is

Exceptions Raised and Exception Handling:

If an employee attempts to apply coupon to a check that does not exist in database, then an exception occurs. At this point an error message is displayed which notifies the employee that the coupon cannot be applied to the check number requested since the check does not exist. Also, if employee attempts to apply the same coupon twice, an exception occurs. A message is displayed notifying the employee that the same coupon cannot be redeemed twice.

3.7 Class Name: UserInterface

The UserInterface class displays various menus on the screen. The UserInterface class is as follows:

UserInterface
-currentMenu: int
+showEmpMenu() +showMgrMenu() +showOrderMenu() +showPizzaMenu() +showDrinkMenu() +showPayrollMenu() +showScheduleMenu() +showSalesMenu()

Class Name: UserInterface

Attributes:

a) –currentMenu: int

currentMenu is a value of type positive integer. It is a private attribute. It holds the number for a menu so that a menu can be selected and displayed.

Operations:

a) +showEmpMenu()

showEmpMenu() is a public function which does not return any value. When this function is called, the menu for an employee is displayed on the screen so that an employee can clock-in, take order, clock-out, start new check etc.

b) +showMgrMenu()

showMgrMenu() is a public function which does not return any value. When this function is called, the manager menu is displayed on the screen so that a manager can choose among options such that view payroll, view sales, void item etc.

c) +showOrderMenu()

showOrderMenu() is a public function which does not return any value. When this function is called, the order menu for an employee is displayed on the screen so that an employee can add pizza, add beverage, or send order to kitchen.

d) +showPizzaMenu()

showPizzaMenu() is a public function which does not return any value. When this function is called, the pizza menu for an employee is displayed on the screen so that an employee can add more pizza to an existing order, set size, set crust, add/remove toppings of a pizza.

e) +showDrinkMenu()

showDrinkMenu() is a public function which does not return any value. When this function is called, the drink menu for an employee is displayed on the screen so that an employee can set type(alcoholic/non-alcoholic) of a drink, add ingredients to a drink or set price of a drink.

f) +showPayrollMenu()

showPayrollMenu() is a public function which does not return any value. When this function is called, the payroll menu for a manager is displayed on the screen so that a manager can print payroll or reset payroll.

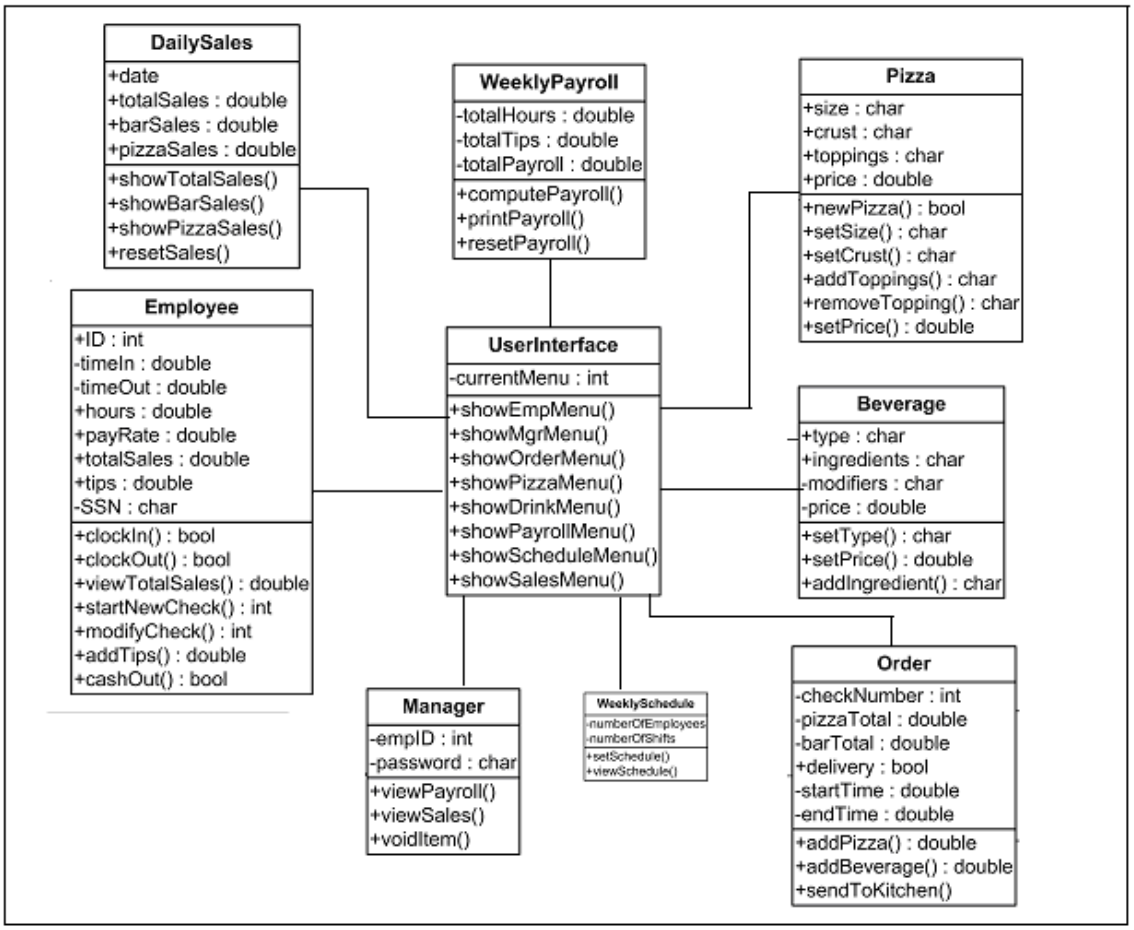
g) +showScheduleMenu()

showScheduleMenu() is a public function which does not return any value. When this function is called, the schedule menu for a manager is displayed on the screen so that a manager can set weekly schedule for employees or view an existing schedule.

h) +showSalesMenu()

showSalesMenu() is a public function which does not return any value. When this function is called, the sales menu for a manager is displayed on the screen so that a manager can view pizza sales, bar sales, total sales, or reset all sales.

Dependencies with Other Classes and Packages:



As we can see in the figure above, the UserInterface class interacts Employee class, Manager class, Order class, Pizza class, Beverage class, WeeklyPayroll class, WeeklySchedule class, and DailySales class. It interacts with all the packages including

Ordering package, Reporting package, RestaurantMaintenance package, and DataInterface package. The mail task of UserInterface class is to provide Graphical User Interface for each menu. Whenever flow of control moves to a class, the class uses a menu from the UserInterface class.

Exceptions Raised and Exception Handling:

If a menu is not available for a class and if that class tries to display a menu, then an exception occurs. At this point, the UserInterface class displays an error message suggesting that no menu exists for this class.

3.8 Check Class

The check Class will allow employee to start a new check, view a check, add items (modify) an existing check, and if the customer had finished ordering then print the check. The Check class is as follows:

Check
+checkNumber : int -empNum : int +tableNumber : int -totalAmount : double -pickUp : bool -delivery : bool -dineIn : bool
+newCheck() : int +addOrder() : int +displayCheck() +printCheck() +splitCheck() : int +applyCoupon() : double +closeCheck() : double +voidCheck() : double

Class Name: Check

Attributes:

a) *+checkNumber : int*

checkNumber is a value identifying the check of type positive integer. It is a public attribute. Check will not have a name, but it will only have a number. Each check will have a unique number until the end of the day, when total sales are figured, and check numbers are cleared for the next day.

b) *–emp_Number: int*

the employee number field is the same as the employee ID from the Employee class. This is in order to match checks with the employee that served the food.

c) *+tableNumber: int*

tableNumber is a value that identifies the table that the order will be taken from. It is a type of positive integer. It is a public attribute. Each table has a number associated with it permanently. In the case of joined tables (such as for a large party), any of the subsumed table numbers may be used.

d) *–totalAmount: double*

totalAmount is the total amount the customer has to pay. It is of type double integer. It is a private attribute. All the food and drink prices that the customer had ordered will be added and saved in this attribute.

e) *-pickUp : bool*

pickup is a value that shows whether the order is a pickup or not. It is of type bool. It is a private attribute. It is initialized when the check class has been created.

g) *-delivery : bool*

delivery is a value that shows whether the order is a delivery order or not. . It is of type bool. It is a private attribute. It is initialized when the check class has been created

h) *dineIn: bool*

dineIn is a value that shows whether the order is a dineIn order or not. . It is of type bool. It is a private attribute. It is initialized when the check class has been created.

0perations:

a) *+newCheck(): int*

newCheck() is a public function which returns a value of type int. When this function is called, a new check will be created and displayed.

b) *+addOrder(): int*

newCheck() is a public function which returns a value of type int. When this function is called, a new check will be created and displayed.

c) *+displayCheck()*

displayCheck() is a public function which does not return any value. When this function is called, the check is displayed according to the check number given.

d) *+printCheck()*

printCheck() is a public function which does not return any value. When this function is called, the check is printed according to the check number given.

e) *+splitCheck(): int*

splitCheck() is a public function which returns a value of type int. When this function is called, the totalAmount is divided among the number of customers.

f) *+applyCoupon(): double*

applyCoupon() is a public function which returns a value of type double. When this function is called, the coupon is used for the check and the amount of the money on the coupon will be deducted from the check.

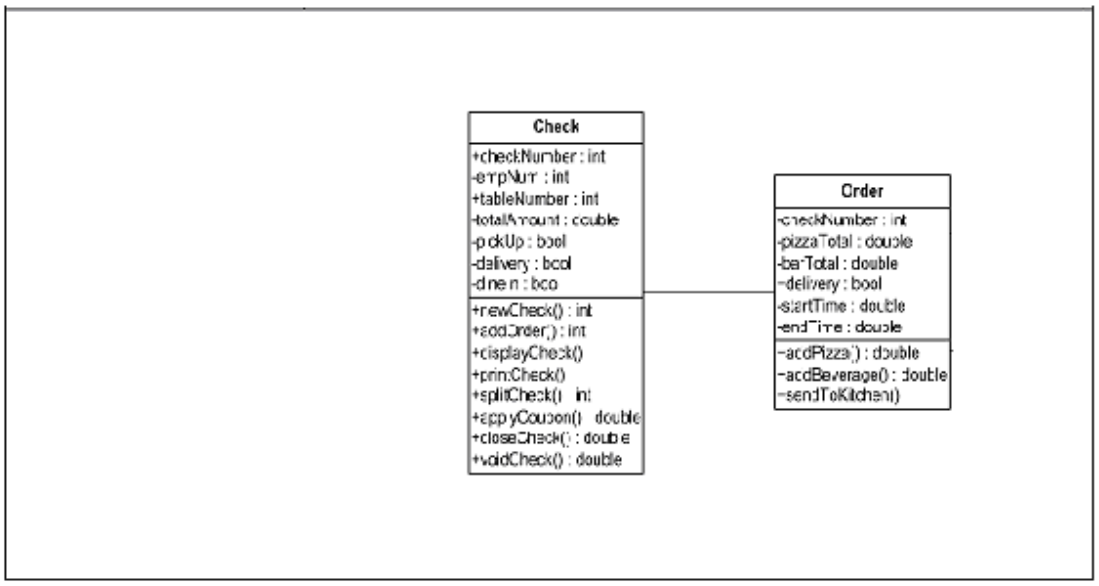
g) *+closeCheck(): double*

closeCheck() is a public function that returns a value of type double. When this function is called, the check is closed and the check is stored in the database. Once the check is closed it is gone and can never be reopened, except for total sales reporting.

h) *+voidCheck(): double*

voidCheck() is a public function which returns a value of type double. When this function is called, the check is erased and nothing is stored in the database.

Dependencies with Other Classes and Packages:



As we can see in the figure above, the Check class interacts with Order class. The employee class lets employees start a new check, view a check, add items (modify) to an existing check, print the check, display the check, split the check. Once the employee starts a new check, the check will be assigned a 4 digit number. This four digit number will consist of the two digit table number and the last two digits will increment by one according to the last check number. Once the check is created, it will be put into the database. When an order is given on the check class will call the functions in the order class depending which order it is.

After the check is closed, all the information related to the check will be saved to the check database.

Exceptions Raised and Exception Handling:

There are several exceptions that can be raised by the check class. For example, if the employee tried to print/open/split or close a nonexistent check (by entering the wrong check number) an exception is raised. To handle this exception, an error message is displayed on the screen, and employee is asked to re-enter the check number into the system.

Also, if the employee tries to apply the same coupon, which has been used before for another order, then an exception is raised. To handle this exception, an error message is displayed on the screen, and employee is asked to re-enter the coupon number into the system.

Another exception is if an employee tries to void a check, then an exception is raised. To handle this exception, an error message is displayed on the screen, and employee is asked to call the manager to void a check. Because a check could only be voided when a manager enters his or her password.

3.9 Pizza Class

The Pizza class defines the type of the pizza that is chosen. The Pizza class is as follows:

Pizza
+size : char -crust : char +toppings : char -price : double
+newPizza() : bool +setSize() : char +setCrust() : char +addTopping() : char +removeTopping() : char +setPrice : double

Class Name: Pizza

Attributes:

a) *+size : char*

size is a name that identifies the pizza size of type char. It is a public attribute. There will be 3 sizes for a pizza. These are

- small
- medium
- large

b) *+crust: char*

crust is a name that identifies the crust type of the pizza. It is of type char. It is a public attribute. There will be 3 crust types available. These are

- regular
- thin
- pan

c) *+toppings : char*

toppings is a name that identifies the toppings to be placed on the pizza. It is of type char. It is a public attribute. There will be 10 toppings available for a Pizza. These are:

- extra cheese
- pepperoni

- black olives
- green olives
- green pepper
- red pepper
- pineapple
- anchovies
- sausage
- hamburger

d) *–price: double*

price is the value the customer has to pay for the ordered pizza. It is of type double integer. It is a private attribute. All the pizza prices that the customer had ordered will be added and saved in this attribute.

Operations:

a) *+newPizza(): bool*

newPizza() is a public function which returns a value of type bool. When this function is called, it creates a new Pizza object so employee can order multiple Pizzas at a time.

b) *+setSize(): char*

setSize() is a public function which returns a value of type char. When this function is called, it asks the employee which size of Pizza is ordered and stores the value into the size attribute.

c) *+setCrust(): char*

setCrust() is a public function which returns a value of type char. When this function is called, it asks the employee which type of crust is ordered and stores the value into the crust attribute.

d) *+addToppings(): char*

addToppings() is a public function which returns a value of type char. When this function is called, it asks the employee which toppings are ordered and stores the value into the toppings attribute.

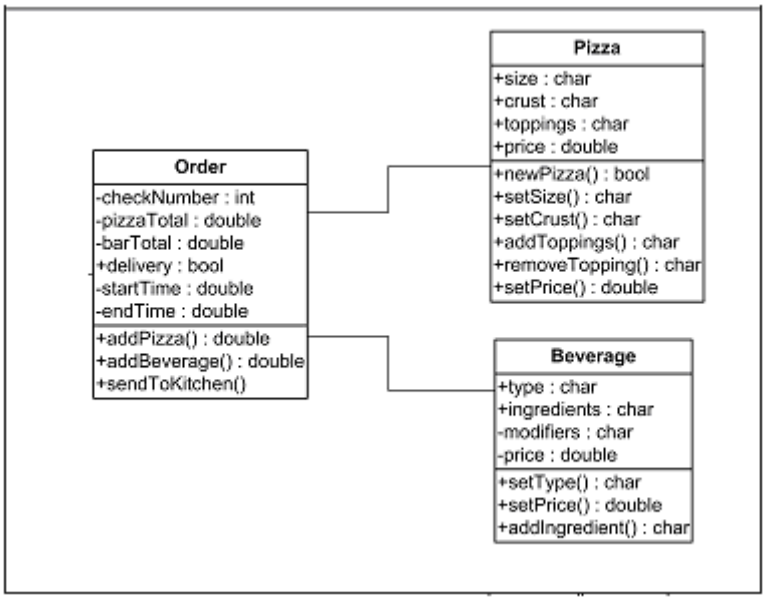
e) *+removeTopping(): char*

removeTopping() is a public function which returns a value of type char. When this function is called, it asks the employee which topping is to be removed and stores the value into the toppings attribute.

f) *+setPrice(): double*

setPrice() is a public function which returns a value of type double. When this function is called, it calculates the price of the Pizza according to the order given.

Dependencies with Other Classes and Packages:



As we can see in the figure above, the Pizza class interacts with the ordering class. The Pizza class lets employees to create a new Pizza and choose the size, crust type, add/ remove toppings of the Pizza. Once a new Pizza is created the Price of the Pizza will be computed according to its, toppings, crust and size.

Exceptions Raised and Exception Handling:

There is an exception that can be raised by the pizza class. For example, if the employee tries to remove a toping that has not been added to the Pizza an exception is raised. To handle this exception, an error message is displayed on the screen, and employee is asked to re-enter the toping that he/she wants to remove.

3.10 Beverage Class

The Beverage class keeps the type of the beverages that are ordered. The Beverage class is as follows:

Beverage
+type : char -ingredients : char +modifiers : char -price : double
+setType() : char +setPrice() : double +addIngredient() : char

Class Name: Beverage

Attributes:

a) *+type : char*

type is a name that identifies what kind of beverage has been ordered. It is a public attribute. There will be three types of beverages. These three beverages divide into subtypes among themselves too. They are listed below

- Beer:
  - Heineken
  - Budweiser
  - Coors
- Alcohol:
  - Absolute
  - Bacardi
  - Cuervo
  - Jack Daniels
- Soft Drinks:
  - Coke
  - Sprite
  - Ginger ale
  -
  -

b) *+ingredients : char*

ingredients is a value that holds the ingredients to be added to the ordered drink. It is of type char. It is a public attribute. Some of these are listed below:

- add coke
- add sprite
- grenadine
- orange juice
- etc.

c) - *modifiers : string*

modifiers is a way for customers to modify their drink selection to their choosing. Some possible modifications are:

- with lemon
- with salt
- chilled mug
- no ice
- double

These are not usually considered ingredients, but rather differences in the way the drink is served. It is of type string. It is a private attribute.

d) *–price: double*

price is the value the customer has to pay for the ordered beverages. It is of type double integer. It is a private attribute. All the beverage prices that the customer had ordered will be added and saved in this attribute.

Operations:

a) *+setType(): char*

setTyp*e*() is a public function which returns a value of type char. When this function is called, it asks the employee which type of beverage is ordered and stores the value into the type attribute.

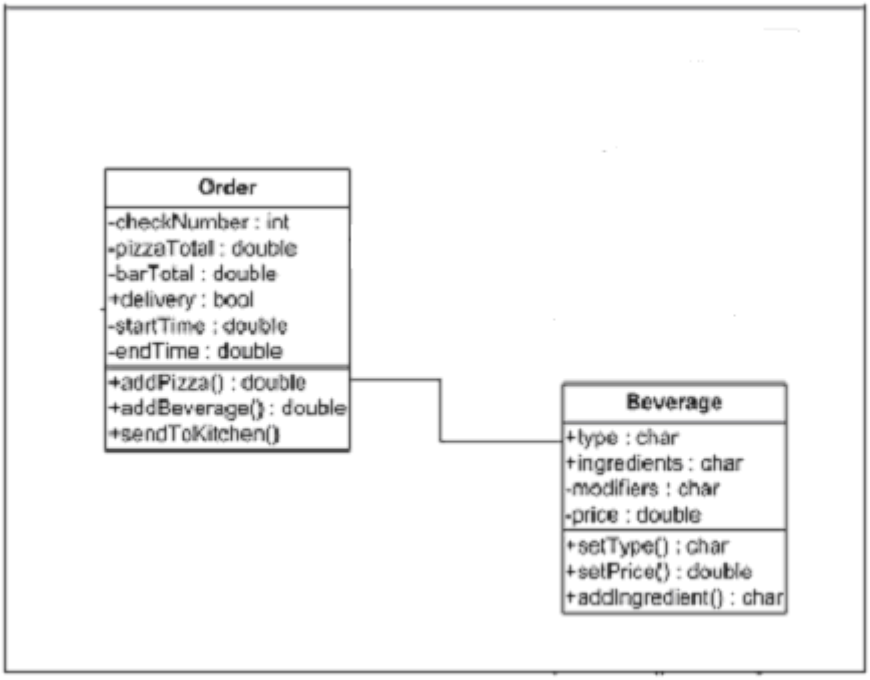
b) *+setPrice(): double*

setPrice() is a public function which returns a value of type double. When this function is called, it calculates the price of the ordered beverages and stores the value into the price attribute.

c) *+addIngredient(): char*

addIngredient() is a public function which returns a value of type char. When this function is called, it asks the employee which type of ingredient is ordered and stores the value into the ingredients attribute.

Dependencies with Other Classes and Packages:



As we can see in the figure above, the Beverage class interacts with the ordering class. The Beverage class lets employees to create a new Beverage and choose the type and add ingredients to it. Once a new Beverage is created the Price of the Beverage will be computed according to its ingredients and type.

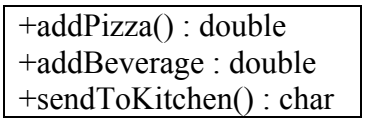
Exceptions Raised and Exception Handling:

All exceptions will be handled by the subroutines called. Since this program is being written in C++ instead of Java, we will add error handling to the classes, methods, and subroutines themselves instead of throwing separate exceptions. For example, if only alphanumeric input is expected in an input, the system will handle non-compliant input with re-prompting, such as “That input was invalid. Only alphanumeric input is acceptable in this menu.” The user will be asked to re-input any erroneous values correctly.

3.11 Order Class

The Order class keeps the orders. It divides the orders into two and keeps them in different classes that are Pizza and Beverage classes.

Order
-checkNumber : int -pizzaTotal : double -barTotal: double



Class Name: Order

Attributes:

a) -*checkNumber : int*

checkNumber is a value that identifies the check. It is of type int. It is a private attribute. It will be a positive integer number.

b) *–pizzaTotal : double*

pizzaTotal is a value that holds the price of the pizzas that had been ordered. It is of type double. It is a private attribute.

c) - *barTotal : char*

barTotal is a value that holds the price of the beverages that had been ordered. It is of type double. It is a private attribute.

Operations:

a) *+addPizza(): double*

addPizza() is a public function which returns a value of type double. When this function is called, a pizza is added to the order, and the price of the pizza is returned.

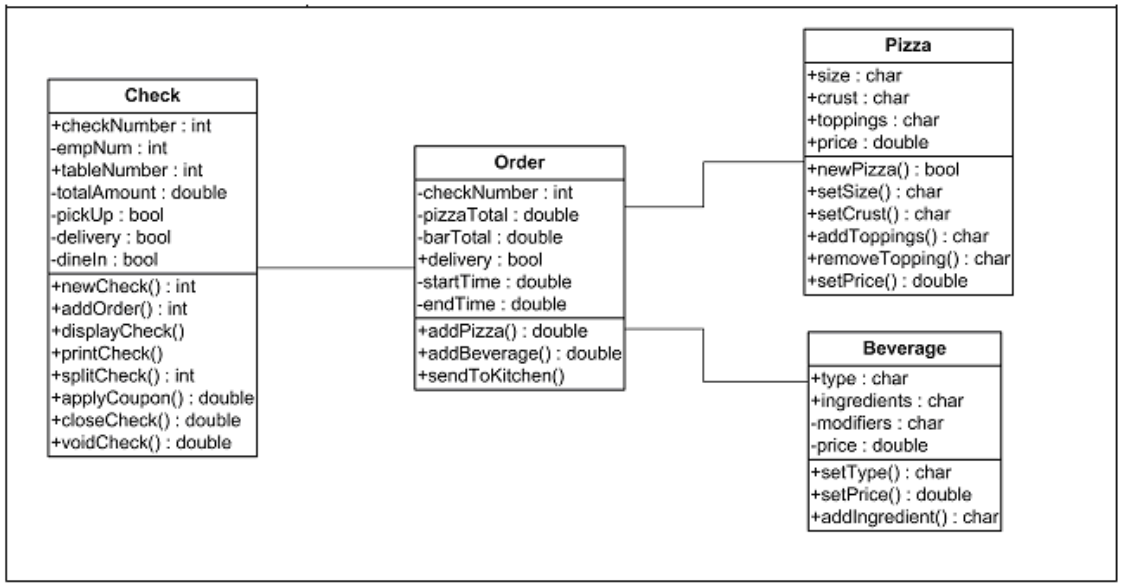
b) *+addBeverage(): double*

addBeverage, just like addPizza, adds an item to the order and returns the price of the added item.

c) *+sendToKitchen()*

sendToKitchen() is a public function which doesn't return any value. When this function is called, the orders on the check is sent to the kitchen to be prepared.

Dependencies with Other Classes and Packages:



As we can see in the figure above, the Order class interacts with the Check class and Pizza class, and with the Beverage class. The order class lets employees add a Pizza/Pizzas or a beverage/beverages. Then the employee can send the order to the kitchen after it is finalized. Once the employee sends the order to the kitchen, the employee can resend the order to the kitchen if the customer modifies it or adds new items. The start time and end time of the order is to see how much time it takes for an order to be ready.

Exceptions Raised and Exception Handling:

There is an exception that can be raised by the Order class. For example, if the check number that is sent to the kitchen is incorrect an exception is raised. To handle this exception, an error message is displayed on the screen, and employee is asked to re-enter the check number into the system.

3.12 DataInterface Class

The DataInterface class keeps track of the database. The class is as follows:

DataInterface
-databaseType : int
+totalNumbers() : double +reportNumbers() : double +findItem() : double +addItem() : double +deleteItem() : double

Class Name: DataInterface

Attributes:

a) -*databaseType : int*

databaseType is a value that holds the type of the database. It is of type integer. It is a private attribute.

Operations:

a) + *totalNumbers() : double*

totalNumbers() is a public function which returns a value of type double. When this function is called, it returns the total of the items listed in that particular database, such as sales for the day, hours for payroll, etc.

b) + *reportNumbers() : double*

reportNumbers() is a public function which returns a value of type double. When this function is called, the system outputs a summary of all the numbers requested (such as individual sales, hours by employee, etc.).

c) +*findItem () : double*

findItem() is a public function which returns a value of type double. When this function is called, it finds the desired item by searching the related database.

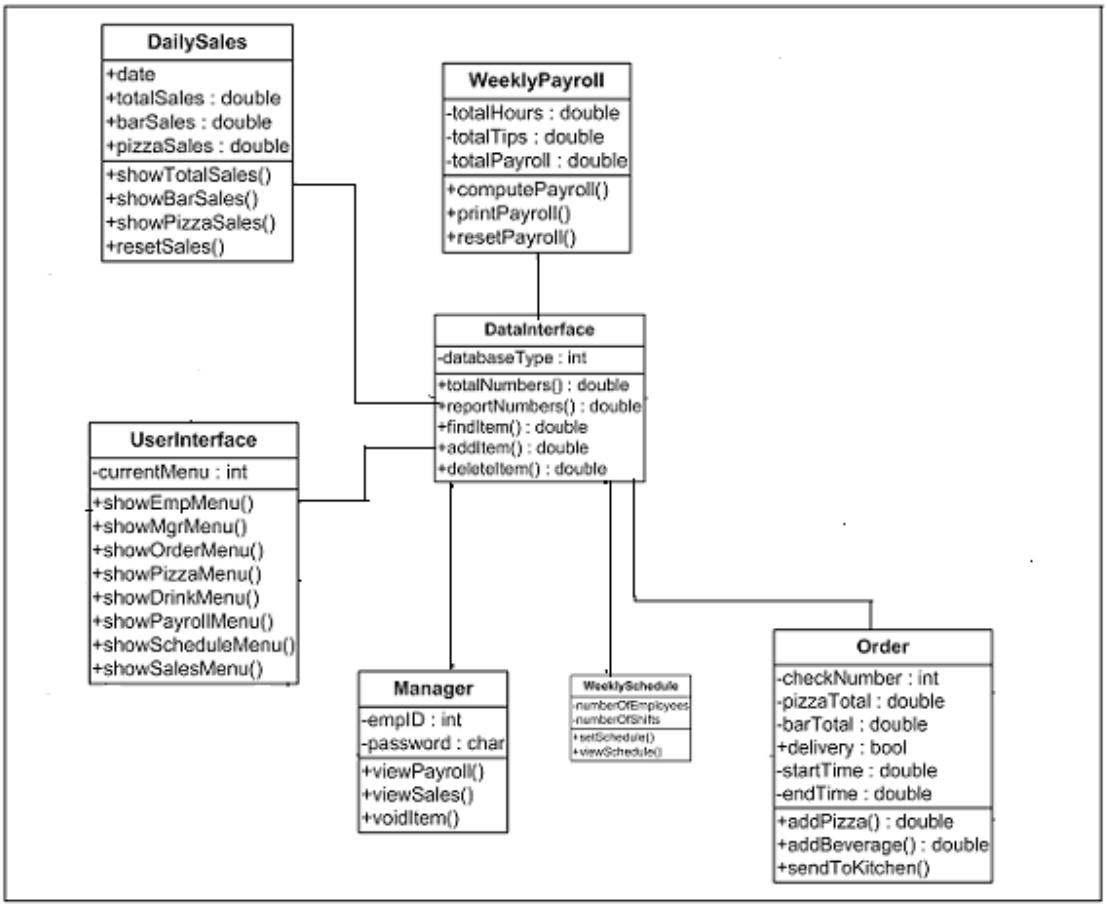
d) +*addItem() : double*

addItem() is a public function which returns a value of type double. When this function is called, it adds an item to the related database.

e) +*deleteItem(): double*

deleteItem() is a public function which returns a value of type double. When this function is called, it deleted the desired item from the database.

Dependencies with Other Classes and Packages:



As we can see in the figure above, the Employee class interacts with DataInterface class and Manager class, and with ordering package. The employee class lets employees clock-in and clock-out. Once the employee chooses to clock-in, he or she is asked to enter his or her employee number. Once the employee is logged-in to the system with proper and approved identification he or she can access the Ordering package and will be able to assist customers with their orders. When the shift of an employee ends, he or she has to clock-out. At the time of clock-out, the employee enters his or her employee number and the amount of tips earned into the system. After cashing out with the manager, the employee's sales for the day will be totaled to the system, and the employee's information will be saved to the payroll database.

Exceptions Raised and Exception Handling:

There are several exceptions that can be raised by the employee class. For example, if the employee identification number is incorrect or if it does not match with the password, an exception is raised. To handle this exception, an error message is displayed on the screen, and employee is asked to re-login into the system.

3.13 FoodMenu Class

The FoodMenu class keeps the food types that the system provides. The class is as follows:

FoodMenu
-crusts : char -toppings : char -sizes: char -specials: char
+addItem() : bool +deleteItem() : bool +setPrice() : double

Class Name: FoodMenu

Attributes:

a) *-crusts: char*

crusts is a value that identifies the new crust type that can be added to the system by the manager. It is of type char. It is a private attribute.

b) *–toppings : char*

toppings is a value that identifies the new toppings type that can be added to the system by the manager. It is of type char. It is a private attribute.

c) - *sizes : char*

sizes is a value that identifies the new size of a Pizza that can be added to the system by the manager. It is of type char. It is a private attribute.

d) - *specials : char*

specials is a value that identifies the new specials that can be added to the system by the manager. It is of type char. It is a private attribute.

Operations:

a) *+addItem()*

addItem() is a public function with no return value. When this function is called, it will ask the manager what item will be added to the system.

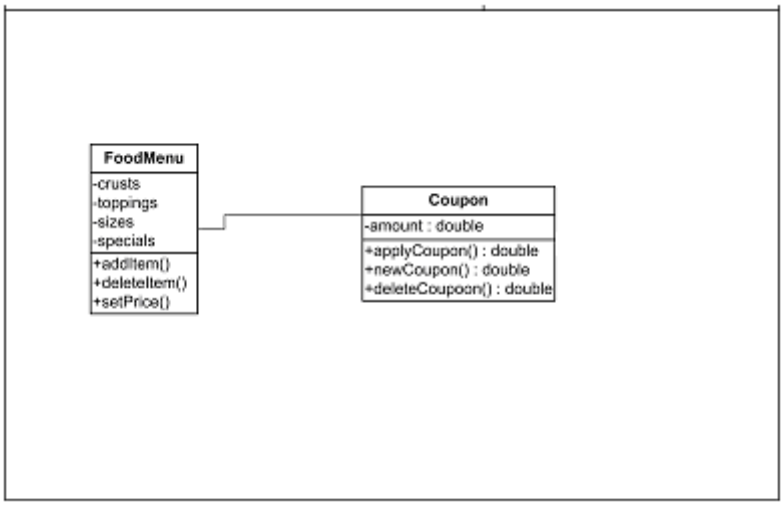
b) *+deleteItem()*

deleteItem() is a public function with no return type. When this function is called, it will ask the manager what item will be deleted from the system.

c) *+setPrice()*

setPrice() is a public function. When this function is called, it will ask the manager, how much price will be set for the item.

Dependencies with Other Classes and Packages:



As we can see in the figure above, the Food Menu class interacts with the coupon class. This is to ensure that any item entered into a coupon actually exists on the Food Menu.

The Food Menu class lets managers to add or delete an item from the menu. Also the manager can set the price of an existing or the new item.

Exceptions Raised and Exception Handling:

As mentioned above, each method or function will handle its own exceptions, including erroneous input by the user. Usually this will be done by re-prompting for correct or acceptable input.

3.14 DrinkMenu Class

The DrinkMenu class keeps the beverage types that the system provides. The class is as follows:

DrinkMenu
-ingredientsAvailable

+addItem() : bool +deleteItem() : bool +setPrice() : double

Class Name: DrinkMenu

Attributes:

a) -*ingredientsAvailable: bool*

ingredientsAvailable is a value that shows whether the system has that ingredient or not. It is of type bool. It is a private attribute.

Operations:

a) +*addItem()*

addItem() is a public function. When this function is called, it will ask the manager what item will be added to the system.

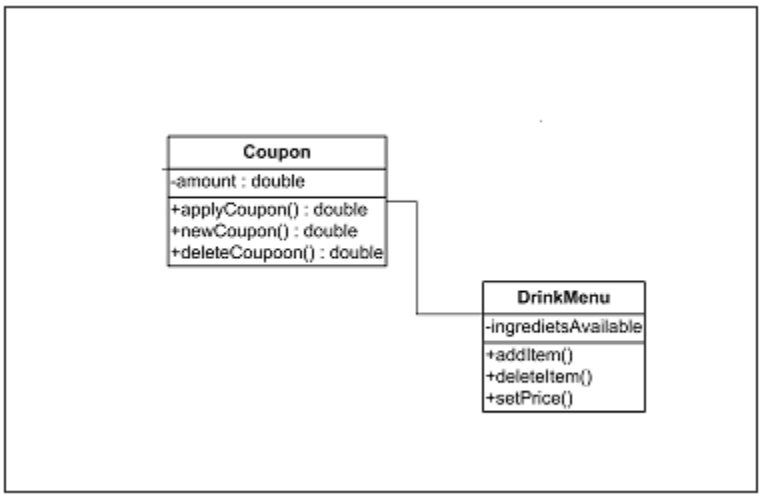
b) +*deleteItem()*

deleteItem() is a public function. When this function is called, it will ask the manager what item will be deleted from the system.

c) +*setPrice()*

setPrice() is a public function which returns a value of type double. When this function is called, it will ask the manager, how much price will be set for the item.

Dependencies with Other Classes and Packages:



As we can see in the figure above, the Drink Menu class interacts with the coupon class. Again, this is so that if a coupon is entered for a drink item (such as a free 2-liter coke with the purchase of two large pizzas), the system will check to verify that the beverage item is on the menu.

The Drink Menu class lets managers to add or delete an item from the menu. Also the manager can set the price of an existing or the new item.

Exceptions Raised and Exception Handling:

Once again, exceptions will be handled in the form of re-prompting in the case of erroneous input.

4. Glossary

Beverage Type, Ingredients, and Modifiers: Beverages may be alcoholic or soft (the two types), and can have multiple ingredients (such as vodka with orange juice and grenadine). Modifiers are text fields like “no ice”, “blended”, “with lemon”, etc., that denote a special customer preference for their beverage.

Cash Out: At the end of a shift, the employee uses the cashOut() option, which totals up all the sales the employee is responsible for in that shift so that the employee can give the restaurant the money for that shift’s orders. There is usually also a “tip out”, where by the employee gives a percentage of sales to the bartender and/or wait staff for their assistance.

Check Number: When the employee starts a new check, the software assigns a new number to that check (the first check of the day is #001, and it increments throughout the day). Whenever the employee wishes to add additional orders (dessert, additional pizza or drinks) to a check, the employee enters the check number in order to access that check directly.

Clock-in: The action that the employee has to do in order to start his/her shift. By clocking in, employee enters his shift starting time.

Clock-out: The action that the employee has to do in order to finish his/her shift. By clocking-out, employee enters his shift ending time and tips made during that shift.

Employee Number: A unique 4-digit identifier used by an employee any time he/she enters an order, clocks in, etc. It is only 4 digits long so that it can be entered quickly.

End of day: End of day report is the report, which prints out all the sales that are done for that day. After end of day is run all sales are printed and then reset.

Finish: “Finish” is an option in the menu where all the orders of a customer are taken and ready to be sent to kitchen or bar. By choosing “Finish” the software updates the check file and sends the orders to the appropriate place. (Kitchen or Bar)

Graphical User Interface (GUI): A GUI is a graphical (rather than purely textual) user interface to a computer.

Kitchen: Every time an employee enters an order, the result is printed and “sent to the kitchen”. The kitchen is ticket-based, and when an order is fulfilled, the cook attaches the printed slip to the pizza or other item completed and sends it via the employee or a helper to the table number recorded on the order.

Manager ID and Password: Similar to employees, managers have unique ID’s that allow them to access the system. However, managers also possess passwords – 6 or more

characters in length – which allow them access to protected functions like payroll, scheduling, voids, etc.

Menu: The user-interface where the restaurant's food and beverages are listed.

Order: Order is the list of food and beverages requested by the customer.

Payroll Hours: Payroll hours are the hours that the employee had worked in the restaurant.

Payroll: Payroll is the money to be paid to the employee.

Pizza Size, Crust, and Toppings: These data members refer to the size (small, medium, large), the crust type (regular hand-tossed, deep pan, thin crust), and toppings (from Anchovies to Zucchini) to be added to a pizza. This includes special "dessert" pizzas guests may order at the end of their meal.

Shift: The time of day employee working. In this Restaurant the two shifts are lunch and dinner.

Submenu: Submenu is the items listed under a particular field selected from the main menu.

Table Number: Each table in the restaurant is numbered, 1-34. An employee will usually be assigned 4-5 tables. The table number is entered in the check so that a helper can deliver a meal without the employee's guidance, freeing the employee to take other orders. A group of tables can be specified by any table in the group for big parties (i.e. a group takes tables 12, 14, and 15 – any of the three can be entered as the table number for that check). Additionally, a table number of 0 is assigned for pick-up or delivery orders.

Timekeeping: Timekeeping is a function to keep track of each employee's working hours.

Tips: At the end of the shift, an employee counts up the tips he or she has earned and adds it to their record so that taxes can be withheld correctly by the payroll software.

Total sales: Total sales are the sales of the restaurant for one day.

User manual: A document describing the user interface of the system such that a user of the software can use it.

Void: "Void" is a function which only the manager can access. Void means that an order has been entered in error to the system and the "Finish" option has already been chosen. In this case, it should be removed from the check and the customer not charged.