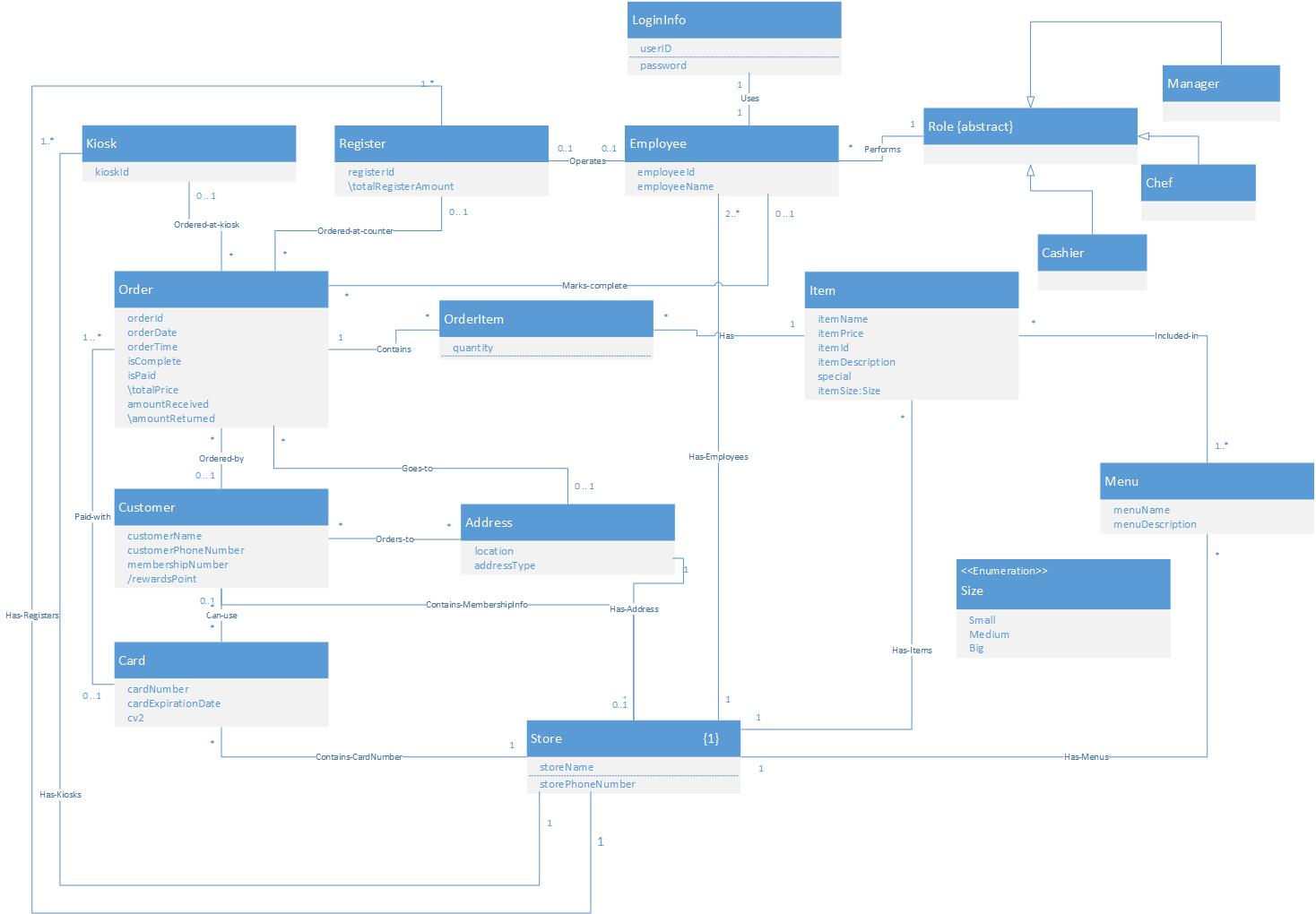
This document contains a model of the domain concepts in a Pizza Ordering System, expressed as a UML class diagram. It also includes a glossary of terms defining each concept described in the UML Class model

A3: DOMAIN MODELING

Team Project Group CS414E

Shaikh Shawon Arefin Shimon, Nathan Lighthart, Yu Qiu, Caleb Tebbe



**GLOSSARY**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Description** | **Author** |
| Inception Draft  (Ver. 1.1) | Sep 29,2014 | First draft. Will be refined in elaboration step for assignment A4 and A5. Contains definition of each concept in domain model of A3. | Shaikh Shawon Arefin Shimon  Caleb Tebbe  Nathan Lighthart  Yu Qiu |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Definitions:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Term** | **Definition and Information** | **Format** | **Validation Rules** | **Relationship to other elements** | **Aliases** |
| Address | A collection of information that holds the address of a membership holding customer, or the destination of a home-delivery order. |  |  | Each order can have 0 or 1 address ( 1 address if home delivery, 0 otherwise). One address can be associated to zero or multiple orders.   One address can contain multiple people who have the membership of the pizza parlor. Similarly, One membership holding customer can have multiple associated address, like one for home, one for office etc. |  |
| addressType | A field indicating what type of address it is (home, business, etc. ). |  |  |  |  |
| amountReceived | Decimal point value showing the amount in Dollars that is received against the order. | Stores two digit after decimal point. | Same as *totalPrice* if paid with cash. |  |  |
| amountReturned | Derived Decimal point value showing the amount in Dollars that is given as a change back to the customer. |  |  | A derived attribute that is updated based on *amountReceived* and *totalPrice* attributes of that current *Order*. |  |
| Can-use | A membership holding *Customer* can use a card to pay for a particular order. In that case that card number will be associated to that *Customer.* |  |  | 1 *Customer* can have zero or multiple credit/debit card. But Every card can be mapped to either a membership holding customer, or a general customer. |  |
| Card | A composite of elements that contain the card information that a customer uses to pay for order. |  |  |  |  |
| cardExpirationDate | A field in *Card* containing the expiration date of that *Card*. |  | Can’t be Null |  |  |
| cardNumber | A field in *Card* that has a unique Number.   See <http://en.wikipedia.org/wiki/Bank_card_number> for more details. | 16-19 digit number. | First 6 digits are Issuer Identification Number (IIN).  Last digit in sequence is a check digit calculated using Luhn algorithm.   Middle digits (9-12 digits) are individual account identifier.   Can’t be Null |  |  |
| Cashier | Role specifying the type of employee as a Cashier. Overriding abstract class Role. |  |  |  |  |
| Chef | Role specifying the type of employee as a Chef. Overriding abstract class Role. |  |  |  |  |
| Contains | An association between an *Order* and *OrderItem* . A *Order* can contain zero or multiple *OrderItem* , but every *OrderItem* can be traced back to only one particular order. |  |  |  |  |
| Contains-CardNumber | An association between Store and Card. A Store can contain reference of all the cards that have been used to purchase Orders |  |  | One store can contain reference of multiple cards. |  |
| Contains-MembershipInfo | An association between Customer holding a membership, and our pizza store. Store contains a list of all membership holding customers. |  |  | One store can contain zero or multiple membership holding customers. |  |
| Customer | A composite of elements storing information about a *Customer* who has a membership of the Pizza parlour. |  |  | A *Order* can contain a reference to a *Customer*, or not. A *Customer* can use multiple cards to give different orders. A *Customer* can have multiple addresses. |  |
| customerName | A field in *Customer* that holds the name of the customer with a membership in the shop. |  | Can’t be Null |  |  |
| customerPhoneNumber | A phone number associated with a *Customer* | 10 digit number | Can’t be Null |  |  |
| cv2 | A field in *Card* used as a security feature for that *Card*. | 3 digit or 4 digit number. | Can’t be Null |  | *Card* Verification Value |
| Employee | An entity which performs action within the Pizza Delivery System. |  |  | Each *Employee* can have exactly one role at one time. Role can be Manager, Cashier, or Chef. Each *Employee* can be associated with 1 *Register* at a time if he has a valid role of Manager or Cashier and he has logged in to that *Register*. Otherwise he will not be associated with any *Register*. |  |
| employeeID | A unique numeric code for each employee. |  | Can’t be Null. |  |  |
| employeeName | A field storing name for an employee, who’s id is stored in the employeeID. |  | Can’t be Null |  |  |
| Goes-to | Association between a particular order and an address. One order can go to only one particular address if it is a home-delivery, or it will not have any associated address if it is an in-house delivery. |  |  |  |  |
| Has | An association between a *OrderItem* and a *Item*. An *OrderItem* objectcan have reference of only one *Item* at a time. But one *Item* can be traced back to zero or multiple *OrderItem* objects. |  |  |  |  |
| Has-Address | An association between Address and Store which indicates that Store has a field of type address that contains the address of the store. |  |  | One address can be mapped to our pizza store. |  |
| Has-Employees | An association between Store and Employee. |  |  | A pizza store must contain atleast 2 employees, one Cashier and a Chef. It can contain more employees. But every employee can be mapped to only one Pizza Store. |  |
| Has-Items | An association showing that the Store object contains a list of Item objects. |  |  | The store can have zero or multiple items – but each item can be mapped to one particular pizza store.    Each Item object is included in at least one Menu object, who’s reference is also stored in the object of Store. |  |
| Has-Kiosks | An association between Kiosk and Store. |  |  | The Pizza store contains at least one Kiosk. And each kiosk in the system must be mapped to only one Store object. |  |
| Has-Menus | An association showing that the Store object contains a list of Menu objects. |  |  | The store can have zero or multiple Menus – but each item can be mapped to one particular pizza store. |  |
| Has-Register | An association between Store and Register. |  |  | The Pizza store contains atleast one Register. And each Register in the system must be mapped to only one Store object. |  |
| Included-in | An association between *Item* and *Menu* . |  |  | One *Item* can be included in one or multiple *Menu*. Each *Menu* can have zero or more *Item*. |  |
| isComplete | A Boolean field that contains indication if all *OrderItem* contained in an *Order* is ready for serving. At the time of *Order* creation this will be false. |  | Can be only true or false. | An *Employee* with *Role Chef* will be able to access the field and change it. |  |
| isPaid | A boolean field that contains indication of if the order has been paid or not. |  | Can be only *TRUE* or *FALSE*. Can be false only when *Order* was created through a *Register* (home delivery for phone) | If true, and if *Order* was placed through *Register* then it will update associated *Register* ‘s *totalRegisterAmount* by current *Order* ‘s *totalPrice* attribute. |  |
| Item | A composite of elements holding information about an *Item* present in the Pizza Store. |  |  | Each *Item* can be included in one or more *Menu*. At the time when an *Item* is being created, it must be through a menu. But later on the *Item* can be included in other menus also.   Each *OrderItem* object has a reference of exactly one *Item*. But one *Item* can be included in multiple *OrderItem* Object. |  |
| itemDescription | A field (text) in *Item* containing the description of the item. |  |  |  |  |
| itemId | A numeric identifier indicating a unique identifier for an *Item*. |  |  |  |  |
| itemName | A field holding the name of the item. |  |  |  |  |
| itemPrice | Decimal point value showing the amount in Dollars of that current item. | Stores two digit after decimal point. |  |  |  |
| itemSize | A field in *Item* that is of Enumeration type Size. This field in *Item* can have only three possible values: Small, Medium and Big. |  |  |  |  |
| Kiosk | An automated unit that a customer himself can use to place an order. |  |  | An order can be placed either to a *Kiosk* or to a *Register* located at the counter. |  |
| KioskID | A unique numeric code that identifies a *Kiosk*. For internal use of the system. |  | Can’t be Null. |  | *Kiosk* Identifier |
| location | A field in *Address* that contains the location of the *Address*. |  |  |  |  |
| LoginInfo | A composition of elements storing login information for every employee. |  |  |  |  |
| Manager | Role specifying the type of employee as a manager. Overriding abstract class Role. |  |  |  |  |
| Marks-complete | An association between an *Order* and a *Employee*. Only an employee with a designated *Role* of *Chef* will be able to mark an order as complete. |  |  | If an order is marked as complete, then An *Employee* is associated to that order who is a chef. If the *Order* is not marked as complete, then the *Order* is associated with zero employees.  Conversely, an *Employee* can be associated with zero or multiple *Order* if he has a *Role* of *Chef,* zero otherwise. |  |
| membershipNumber | A numeric code that identifies a *Customer* with a membership. |  | Can’t be null. |  |  |
| Menu | A grouping of item based on some common interest. |  |  | One *Item* can be included in one or multiple *Menu*. Each *Menu* can have zero or more *Item*. |  |
| menuDescription | A field in *Menu* that contains a short description of that particular menu or grouping of *Item*s. |  |  |  |  |
| menuName | A field in *Menu* that denotes the name for the *Menu* |  |  |  |  |
| Operates | Association between a particular register and a particular employee. A register can be operated by an *Employee* only if the *Employee* has a role of *Cashier* or *Manager*. |  |  | Every register can be associated with only one employee at a time. An *Order* can be placed at a *Register* only when an employee with a valid role is logged onto it. An employee can be associated with a register at a time, if he has a valid role. Otherwise he will not be associated with any *Register*. |  |
| Order | An order placed by a customer in the Pizza *Order*ing System. |  |  | An *Order* can contain multiple *OrderItem*. An *Order* will contain either a reference to a *Kiosk* or a *Register*. It can be marked as complete by an *Employee* who has a *Role* of *Chef*. An *Order* can be paid via cash or *Card*, and if paid using *Card*, the *Order* receipt should contain the reference of the *Card* number. If the *Order* is a home-delivery, then the *Order* must contain a reference to one particular *Address*. |  |
| orderDate | A field in *Order* containing the date the order was placed. |  | Can’t be Null |  |  |
| Ordered-at-counter | An association between *Register* and *Order* Class. |  |  | Association Relates an *Order* with a *Register* object if the order was passed through a *Register*. Each order can be related to either 0 or 1 *Register*.  Conversely, each register can take 0 or multiple orders. |  |
| Ordered-at-Kiosk | An association between *Kiosk* and *Order* Class. |  |  | Association Relates an *Order* with a *Kiosk* object if the order was passed through a *Kiosk*.  Each order can be related to either 0 or 1 *Kiosk*. Conversely, 0 or multiple orders can be placed through a *Kiosk*. |  |
| Ordered-by | Association between an *Order* and a *Customer* who is a member of the pizza parlor. An *Order* can be placed by a member of the pizza parlor, or a regular customer. |  |  | If an *Order* is placed by a *Customer* having a membership number, his reward points update with each purchase. |  |
| orderID | A numeric code that identifies an *Order*. Usually symbolized with a bar code placed on receipt. |  | Can’t be Null |  |  |
| OrderItem | A composite of elements containing a reference to an *Item*, and the quantity of the item. |  |  |  |  |
| Orders-to | Association between a particular customer and an address. If the customer is a membership holder, he can tell the order to be sent at one of the addresses that is already listed in his account. |  |  |  |  |
| orderTime | A field in *Order* containing the time on the orderDate the order was placed. |  | Can’t be Null |  |  |
| paid with | A relationship between a *Card* (Credit/Debit) and an *Order*. |  |  | If a customer uses a credit card to pay for an *Order*, then a *Card* is associated with an *Order*. 0 otherwise.   Conversely, a *Card* can be used to pay for multiple *Order*, but a *Card* information will be stored in the System only when it has been used for paying for atleast one *Order*. |  |
| Password | An alphanumeric field in *LoginInfo* that contains a corresponding password for the employee to login to the system. |  |  |  |  |
| Performs | An association between Role and Employee that signifies which role the employee performs. |  |  | Each employee can perform only one role at a time. But One role can be mapped to zero or multiple employee at a time. |  |
| quantity | A field in *OrderItem* that contains the number of a particular item ordered in a particular order. |  |  |  |  |
| Register | A unit which is operated by an employee to take an order. Cashbox is integrated with register. |  |  | An order can be placed either to a *Kiosk* or to a *Register* located at the counter.  A *Register* can start taking order only when an *Employee* with a valid role is logged on to the system and mapped with that *Register*. |  |
| registerID | A unique numeric code that identifies a *Kiosk*. For internal use of the system. |  | Can’t be Null. |  | *Register* Identifier |
| rewardsPoint | A numeric value that signifies the amount of benefit a member can avail from the pizza parlor. |  |  | A derived attribute in *Customer* that is updated by using *totalPrice* in *Order* when the attribute *isPaid* is set to TRUE in the related order. Updated based on business rules of the pizza parlor. |  |
| Role | An abstract class containing template for different responsibilities for the employees. The system will specify the access/permission for an employee according to his/her role by overriding operations of this class. |  |  |  |  |
| special | A Boolean value indicating if this item is set as special value for the day. |  |  |  |  |
| Store | A singleton class that stores information about the pizza store, like the name, address, all items, all menus, all membership holding customer information , information about kiosks/Registers/Employees |  |  |  |  |
| storeName | A field in Store class that contains the name of the Store |  |  |  |  |
| storePhoneNumber | A field in Store class that contains the phone number of the store. | 10 digit number | Can’t be Null |  |  |
| totalPrice | A derived attribute in *Order* that contains total price of all *OrderItem* in Decimal point value. Value corresponds to Dollars and cents. Includes local Sale Tax. | Stores two digit after decimal point |  | A derived attribute that can be updated by adding *itemPrice* attribute of each item mapped to each *OrderItem* included in the *Order*, and then applying business logic to include local sale tax with added amount. |  |
| totalRegisterAmount | Derived Decimal point value that stores the total amount of dollars and cents that is stored in the *Register* Cash Box. | Stores two digit after decimal point |  | A derived attribute that can be updated by using attribute totalPrice of each order associated with that *Register*. Will be updated only when *isPaid* attribute in related order is *TRUE*. |  |
| userID | A field in *LoginInfo* that contains a user Identification name (Text) unique to a particular *Employee*. |  |  |  |  |
| Uses | An one to one association between each *Employee*  and a *LoginInfo* |  |  | Each *Employee* has only one *LoginInfo* and Every *LoginInfo* can be mapped back to only one *Employee*. |  |
|  |  |  |  |  |  |